Spass und Software-Entwicklung
Zur Motivation von Open-Source-Programmierern

Dissertation
der Wirtschaftswissenschaftlichen Fakultät
der Universität Zürich

zur Erlangung der Würde
eines Doktors der Ökonomie

vorgelegt von

Benno Luthiger Stoll
von Risch/ZG und Zürich

genehmigt auf Antrag von

Prof. Dr. E. Franck
Prof. Dr. U. Schallberger

Die Wirtschaftswissenschaftliche Fakultät der Universität Zürich gestattet hierdurch die Drucklegung der vorliegenden Dissertation, ohne damit zu den darin ausgesprochenen Anschauungen Stellung zu nehmen.

Zürich, den 8. Februar 2006

Der Dekan: Prof. Dr. H. P. Wehrli

Wenn einer, was er tut, nicht muss, dann tut er es mit Hochgenuss.
(Volksmund)

Programming is the Great Game. It consumes you, body and soul. When you're caught up in it, nothing else matters.
(„How Software Companies Die“, Orson Scott Card)

This is a program for hackers by a hacker. I've enjouyed [sic!] doing it, and somebody might enjoy looking at it and even modifying it for their own needs.
(Ankündigung von Linux v.0.02 durch Linus Torvalds, Oktober 1991)

Dieses Dokument wurde mit LaTeX gesetzt.

Dank

Diese Dissertation wurde möglich, weil die Wirtschaftswissenschaftliche Fakultät der Universität Zürich die Promotion auch solchen wissenschaftlich interessierten Personen ermöglicht, die nicht über einen wirtschaftswissenschaftlichen Studienabschluss verfügen, weil Prof. E. Franck sich als Doktorvater für mein Dissertationsvorhaben zur Verfügung stellte und weil PD Dr. C. Jungwirth einen wesentlichen Teil der Betreuungsarbeit übernahm. Diesen Personen und Gremien gehört mein erster Dank in diesem Zusammenhang.

Ein grosses Dankeschön gehört meiner Frau und meinem Kind, welche mich in keinem Zeitpunkt während der Arbeit an dieser Dissertation vergessen liessen, dass es noch andere wichtige Sachen bzw. wichtige andere Sachen gibt neben der Forschung und der Arbeit. Ein ähnlich gelagertes Dankeschön geht an meine Kollegen in der Saunagruppe „Kommando R.G.“, welche mich nie vergessen liessen, dass es neben der Wissenschaft beispielsweise noch die Sauna gibt. Innerhalb dieser Saunagruppe geht ein spezieller Dank an K. Frei, der mir mehrmals grosszügig sein Ferienhaus zur Verfügung stellte, damit ich mich in Ruhe auf meine wissenschaftliche Arbeit konzentrieren konnte, was mein Vorhaben jeweils grosse Stücke weiterbrachte.

Herzlichen Dank auch an Prof. U. Backes-Gellner, Prof. M. Osterloh, Dr. T. Pudack, Dr. P. Moog und J. Mure, welche mir wertvolle Tipps zukommen liessen, R. Mazzoni und J. Forrer von den ID der Universität Zürich sowie der Sektion Technologie und Informationsmanagement der ETH Zürich für die Benützung der IT-Infrastruktur, welche für die Online-Umfrage benötigt wurde, sowie den Pretester/innen für die sorgfältigen Rückmeldungen in der Testphase der Online-Umfrage.

Das letzte Dankeschön geht natürlich an die Software-Entwickler/innen, die an der Umfrage teilnahmen und damit mir und den Leser/innen dieser Arbeit einen interessanten Einblick in das Phänomen „Open Source“ ermöglichten, sowie den Open-Source-Programmierer/innen überhaupt. Ihr kreativer und generöser Einsatz macht die Welt wenn nicht besser, so doch sicher interessanter.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Forschungsfragen . . . 2
- 1.2 Zur Relevanz der Forschungsfragen . . . 3
- 1.3 Gliederung . . . 4

I Theorie **7**

2 Theoretisches Konzept **9**
- 2.1 Modell . . . 9
- 2.2 Operationalisierung . . . 10
- 2.3 Der Stand der Forschung . . . 11
  - 2.3.1 Flow . . . 12
  - 2.3.2 Online-Umfragen . . . 12

3 Das Phänomen „Open Source“ **13**
- 3.1 Open-Source-Software als öffentliches Gut . . . 13
- 3.2 Urheberrecht und Lizenz . . . 14
- 3.3 Das Open-Source-Entwicklungsmodell . . . 16
- 3.4 Die Sicht der Benutzer-Programmierer . . . 17
- 3.5 Zur Motivation von Open-Source-Programmierern . . . 18
  - 3.5.1 Gebrauch . . . 18
  - 3.5.2 Reputation und Signalproduktion . . . 19
  - 3.5.3 Identifikation mit der Gruppe . . . 19
  - 3.5.4 Lernen . . . 20
  - 3.5.5 Altruismus . . . 20
- 3.6 Spass und Open Source . . . 20
- 3.7 Der Stand der Forschung . . . 23

4 Das Flow-Konzept **39**
- 4.1 Spass und Flow . . . 39
- 4.2 Flow am Computer . . . 44
- 4.3 Flow messen . . . 45

5 Methodologie der Online-Umfrage **49**
- 5.1 Die Fragebogen-Methode . . . 49
- 5.2 Fragebogenuntersuchungen im Internet . . . 52

5.3 Rücklaufquote und Rücklaufgeschwindigkeit 53

6 Forschungsdesign 57
6.1 Aufbau des Fragebogens 57
6.2 Programmierung des Fragebogens 60
6.3 Pretest 61
6.4 Auswahl der Open-Source-Plattformen 63
6.5 Durchführung der Umfrage 65

II Forschungsresultate 67

7 Open-Source-Umfrage 69
7.1 Deskriptive Auswertungen 69
7.1.1 Allgemeines Antwortverhalten 69
7.1.2 Demographische Angaben 71
7.1.3 Nationalität 74
7.1.4 Zeitliches Engagement 78
7.1.5 Flow-Erleben und Open Source 81
7.2 Engagement, Einsatzbereitschaft und Produktivität 86
7.2.1 Engagement 86
7.2.2 Einsatzbereitschaft 92
7.2.3 Output und Produktivität 96
7.3 Flow-Faktoren 101
7.3.1 Faktorenanalyse der Flow-Items 101
7.3.2 Bedingungen für Flow-Erleben 105
7.4 Typisierung 113
7.4.1 Clusteranalyse 113
7.4.2 Flow-Empfinden der unterschiedlichen Typen 120
7.4.3 Engagement der unterschiedlichen Typen 125
7.4.4 Einsatzbereitschaft der unterschiedlichen Typen 127
7.4.5 Einschätzung der Projektarbeit 127
7.4.6 Weitere typenspezifische Mermale 131
7.5 Die Bedeutung von Spass 136
7.5.1 Das Modell 136
7.5.2 Einsatzbereitschaft als Funktion von Spass 137
7.5.3 Engagement als Funktion von Spass 141
7.5.4 Die Bedeutung von Spass bei Professionals 142
7.6 Interpretation 144

8 Befragung der kommerziellen Software-Entwickler 147
8.1 Deskriptive Auswertungen 147
8.1.1 Allgemeines Antwortverhalten 147
8.1.2 Demographische Angaben 148
8.1.3 Arbeitsplatz und Projektarbeit 150
8.2 Flow und Engagement bei kommerziellen Programmierern 154
8.2.1 Faktorenanalyse der Flow-Items 154

8.2.2 Spass und Engagement am Arbeitsplatz 157
8.3 Korrelationen 159
8.3.1 Bedingungen für Flow-Erleben 159
8.3.2 Weitere Korrelationen 163
8.4 Typisierung 168
8.4.1 Clusteranalysen 168
8.4.2 Verifikation der Typisierungen 172
8.4.3 Zusammenhang mit Flow-Empfinden 173
8.5 Folgerungen 178

9 Vergleich der Open-Source- und kommerziellen Entwickler 183
9.1 Alter und Geschlecht 183
9.2 Vergleich der Flow-Komponenten 185
9.3 Vergleich der Projektsituation 191
9.4 Analyse der Unterschiede 194
9.5 Folgerungen 198

10 Schlussfolgerungen 201

A Die Open-Source-Definition 205

B Herleitung der Produktionsfunktion 209

C Der FASD-Fragebogen 213

D Lancierung der FASD-Studie 229

E Bemerkung Zeitberechnung 233

Tabellenverzeichnis

3.1 Komponenten des Open-Source-Entwicklungsmodells 17
3.2 Unterschiede von kommerziellen zu Open-Source-Projekten . . . 22
3.3 Empirische Untersuchungen zu Open Source 37

4.1 Komponenten von Flow (aus Rheinberg, 1997, S. 143) 40
4.2 Bedingungen für das Erleben von Flow gemäss Hoffman und Novak (1996) 41

7.1 Fehlende Werte 69
7.2 Sprache 71
7.3 Verwendete Open-Source-Plattform 71
7.4 Antwortzeiten 71
7.5 Alter 72
7.6 Anzahl Jahre Programmiererfahrung 72
7.7 Geschlecht 73
7.8 Programmiererfahrung nach Geschlecht 73
7.9 Wohngemeinschaft 73
7.10 Open-Source-Entwickler mit Kindern 73
7.11 Anzahl Hobbys (inklusive Programmieren) 74
7.12 Beschäftigungsgrad 75
7.13 Bisher erreichte Projekt-Rolle 75
7.14 Projekt-Rolle als Funktion der Erfahrung 76
7.15 Nationalität 76
7.16 Relative Häufigkeit 77
7.17 Zeiteinsatz für Open Source (Stunden pro Woche) 78
7.18 Anteil Zeiteinsatz in Freizeit 80
7.19 Häufigkeit von Abgabeterminen 80
7.20 Zeiteinsatz der Typen 81
7.21 Flow-Erleben 82
7.22 Zukünftiges Engagement 83
7.23 Arbeiten in Open-Source-Projekten 83
7.24 Teilnahmemotive als Projektmitglied 84
7.25 Motive als Projektverantwortlicher 85
7.26 Zeiteinsatz als Funktion der Rolle 86
7.27 Zeiteinsatz in Freizeit als Funktion der Rolle 88
7.28 Zeiteinsatz während Arbeitszeit als Funktion der Rolle 89

7.29 Zeiteinsatz in Freizeit als Funktion der Anzahl Hobbys 90
7.30 Zeiteinsatz in Freizeit als Funktion des Beschäftigungsgrads . . . 92
7.31 Einsatzbereitschaft als Funktion der Projekt-Rolle 93
7.32 Output 96
7.33 Output als Funktion der Projekt-Rolle 97
7.34 Produktivität 98
7.35 Produktivität als Funktion der Projekt-Rolle 99
7.36 Kommunalitäten der rotierten Lösung 102
7.37 Interpretation der Faktoren 104
7.38 Flow/Spass als Funktion der Rolle 107
7.39 Korrelation Flow und Zeiteinsatz 107
7.40 Korrelation Flow und Zeiteinsatz in Freizeit 107
7.41 Korrelation Flow und Zeiteinsatz während Arbeitszeit 108
7.42 Korrelation Flow und Einsatzbereitschaft 108
7.43 Korrelation Flow und Einsatzbereitschaft (29) 109
7.44 Korrelation Flow und Einsatzbereitschaft (30) 109
7.45 Korrelation Flow und Einsatzbereitschaft (31) 110
7.46 Korrelation Flow und Output 111
7.47 Motive als Programmierer in einem Open-Source-Projekt 114
7.48 Motive als Projektleiter an einem Open-Source-Projekt 115
7.49 Faktorenanalyse der Motive als Programmierer 116
7.50 Clusterbildung mit den Motivationsfaktoren der Programmierer . . 117
7.51 Kreuztabelle: Typen von Programmierern 117
7.52 Faktorenanalyse der Motive als Projektleiter 118
7.53 Clusterbildung mit den Motivationsfaktoren der Projektleiter . . . 119
7.54 Kreuztabelle: Typen von Projektleitern 119
7.55 Kreuztabelle: Übereinstimmung der Typen 120
7.56 Kreuztabelle: Vergleich mit Open-Source-Typen 121
7.57 Flow-Empfinden der Typen von Programmierern 122
7.58 Flow-Komponenten bei Projektleitern: Signifikanz der Unterschiede 123
7.59 Flow-Empfinden der Typen von Projektleitern 124
7.60 Zeiteinsatz der Typen von Programmierern 126
7.61 Zeiteinsatz von Programmierern: Signifikanz der Unterschiede . . 127
7.62 Zeiteinsatz der Typen von Projektleitern 128
7.63 Zeiteinsatz von Projektleitern: Signifikanz der Unterschiede 129
7.64 Einsatzbereitschaft der unterschiedlichen Typen 130
7.65 Einschätzung der Projektarbeit durch Programmierer 132
7.66 Einschätzung der Projektarbeit durch Projektleiter 133
7.67 Erfahrung der unterschiedlichen Typen 135
7.68 Einsatzbereitschaft als Funktion von Spass 1 137
7.69 Einsatzbereitschaft als Funktion von Spass 2 138
7.70 Einsatzbereitschaft als Funktion von Spass und Zeit 3 139
7.71 Einsatzbereitschaft als Funktion von Spass und Zeit 4 140
7.72 Engagement als Funktion von Spass und Zeit 1 141
7.73 Engagement als Funktion von Spass und Zeit 2 142
7.74 Professionelle Einsatzbereitschaft als Funktion von Spass 143
7.75 Professioneller Freizeiteinsatz als Funktion von Spass 144

8.1 Fehlende Werte 148
8.2 Alter 148
8.3 Anzahl Jahre Programmiererfahrung 149
8.4 Geschlecht 149
8.5 Programmiererfahrung nach Geschlecht 149
8.6 Beschäftigungsgrad 150
8.7 Flow-Erleben 150
8.8 Engagement am Arbeitsplatz 152
8.9 Verhältnis zum Arbeitgeber 152
8.10 Open Source am Arbeitsplatz 152
8.11 Projektsituation 153
8.12 Kommunalitäten der rotierten Lösung 155
8.13 Interpretation der Faktoren 156
8.14 Einsatzbereitschaft als Funktion von Spass 158
8.15 Flow-Empfinden und Engagement am Arbeitsplatz 160
8.16 Flow-Empfinden und Verhältnis zum Arbeitgeber 161
8.17 Flow-Empfinden und Projektarbeit 162
8.18 Flow-Empfinden und Erfahrung 163
8.19 Projektarbeit und Engagement 164
8.20 Projektarbeit und Verhältnis zum Arbeitgeber 165
8.21 Verhältnis zum Arbeitgeber und Engagement 166
8.22 Projektsituation und Erfahrung 167
8.23 Clusteranalyse der Fragen zum Engagement 170
8.24 Clusteranalyse der Fragen zum Verhältnis zum Arbeitgeber 170
8.25 Clusteranalyse der Fragen zur Projektsituation 171
8.26 Rolle im Software-Projekt 171
8.27 Kreuztabelle: Engagement vs. Verhältnis zum Arbeitgeber 172
8.28 Kreuztabelle: Engagement vs. Projektsituation 173
8.29 Kreuztabelle: Engagement vs. Projekt-Rolle 174
8.30 Kreuztabelle: Verhältnis zum Arbeitgeber vs. Projektsituation . . . 174
8.31 Flow-Empfinden in Abhängigkeit des Engagements 176
8.32 Flow-Empfinden und Verhältnis zum Arbeitgeber 177
8.33 Flow-Empfinden in Abhängigkeit der Projektsituation 179

9.1 Altersvergleich 184
9.2 Vergleich der Geschlechter-Verhältnisse 184
9.3 Flow-Empfinden: Vergleich 188
9.4 Flow-Faktoren: Open-Source-Hacker und Professionals 189
9.5 Projektsituation: Vergleich OSS und kommerzielle Umfrage . . . 192
9.6 Projektsituation: Vergleich Hacker und Professionals 192
9.7 Projektvision: Diskrepanz zwischen Soll- und Ist-Zustand 194
9.8 Unterschiede zwischen den Software-Entwicklungsmodellen . . . 195
9.9 Korrelationen: Unterscheidungsmerkmale mit Flow-Faktoren . . . 197

C.1 Fragebogen für Open-Source-Entwickler (englisch) 213
C.2 Fragebogen für Open-Source-Entwickler (deutsch) 217
C.3 Fragebogen für kommerzielle Software-Entwickler (englisch) . . . 221

C.4 Fragebogen für kommerzielle Software-Entwickler (deutsch) . . . 224

Abbildungsverzeichnis

4.1 Konzeptuelles Modell von Novak, Hoffman und Yung (1997, S. 3) 42
4.2 Acht-Kanal-Modell von Flow (aus Novak und Hoffman, 1997, S. 11) 43

7.1 Anteil Zeiteinsatz in Freizeit 79
7.2 Zeiteinsatz als Funktion der Rolle 87
7.3 Zeiteinsatz in Freizeit als Funktion der Rolle 88
7.4 Zeiteinsatz während Arbeitszeit als Funktion der Rolle 89
7.5 Zeiteinsatz in Freizeit als Funktion der Anzahl Hobbys 91
7.6 Zeiteinsatz in Freizeit als Funktion des Beschäftigungsgrads . . . 91
7.7 Einsatzbereitschaft als Funktion der Projekt-Rolle 94
7.8 Einsatzbereitschaft in Abhängigkeit der Anzahl der Hobbys 95
7.9 Einsatzbereitschaft und Beschäftigungsgrad 95
7.10 Output als Funktion der Projekt-Rolle (Patches) 97
7.11 Output als Funktion der Projekt-Rolle (Module etc.) 98
7.12 Produktivität als Funktion der Projekt-Rolle (Klassen etc.) 100
7.13 Screeplot der Faktorenanalyse mit 26 Frage-Items 102
7.14 Flow/Spass als Funktion der Rolle in Open-Source-Projekten . . . 106
7.15 Flow-Empfinden und Mitbewohner 112
7.16 Flow-Empfinden von Programmierern 123
7.17 Flow-Empfinden von Projektleitern 125
7.18 Zeiteinsatz von Programmierern 126
7.19 Zeiteinsatz von Projektleitern 128
7.20 Einsatzbereitschaft von Programmierern 129
7.21 Einsatzbereitschaft von Projektleitern 130
7.22 Einschätzung der Projektarbeit durch Programmierer 131
7.23 Einschätzung der Projektarbeit durch Projektleiter 134

8.1 Screeplot der Faktorenanalyse 155
8.2 Flow-Empfinden in Abhängigkeit des Engagements 175
8.3 Flow-Empfinden und Verhältnis zum Arbeitgeber 176
8.4 Flow-Empfinden in Abhängigkeit der Projekt-Rolle 177
8.5 Flow-Empfinden in Abhängigkeit der Projektsituation 178

9.1 Vergleich der Faktoren aus der OSS-Umfrage 186
9.2 Vergleich der Faktoren aus der kommerziellen Umfrage 187
9.3 Flow-Faktoren: Open-Source-Hacker und Professionals 190
9.4 Projektsituation: Vergleich OSS und kommerzielle Umfrage . . . 191

9.5 Projektsituation: Vergleich Hacker und Professionals 193

B.1 Indifferenzkurven von Arbeitnehmern bei der Wahl zwischen Arbeitszeitsouveränität und Lohn 210
B.2 Zusammenhang zwischen Spass und Engagement 211
B.3 Zusammenhang zwischen Freizeit und Engagement 212

C.1 Screenshot des FASD-Fragebogens 227

Kapitel 1

Einleitung

Open-Source-Software können alle Computer-Benützende legal aus dem Netz herunterladen und auf ihren Computern installieren, *ohne* dass sie dafür den Herstellern irgendetwas bezahlen müssen. Die Open-Source-Bewegung hat in der Zeit seit ihrem Bestehen immer wieder herausragende Produkte hervorgebracht, welche den Vergleich mit kommerziellen Produkten keineswegs scheuen müssen. Die Open-Source-Bewegung hat in den letzten Jahren kräftig an Dynamik gewonnen und stellt für die Anbieter von kommerzieller Software eine ernsthafte Herausforderung dar. Wie ist dieses Phänomen zu erklären? Hat der *Homo oeconomicus* im Softwarebereich den Verstand verloren?

Um diese Fragen beantworten zu können, müssen wir nach den Motiven der Akteure im Open-Source-Bereich fragen. Die wissenschaftliche Forschung, welche sich mit der Untersuchung des Open-Source-Phänomens befasst, hat eine Vielfalt solcher Motive identifizieren und auch theoretisch gut begründen können.[1] Was bezüglich der Motivationsforschung von Open-Source-Entwicklern aussteht, ist eine Quantifizierung der verschiedenen Motive. Wir können zwar die Motivation der Beitragsleister von Open-Source-Projekten theoretisch gut nachvollziehen, wissen aber nicht, wie gross die Bedeutung dieser Motive in der Praxis tatsächlich ist. Möglicherweise machen wir uns ein falsches Bild der Open-Source-Entwickler, weil wir ihnen Motive unterstellen, die theoretisch elegant sind, in der Praxis aber keine Bedeutung haben. Ein realistisches Bild der Akteure im Open-Source-Bereich wird erst möglich, wenn wir neben der Feststellung der *Existenz* von Motiven zum Engagement im Open-Source-Bereich auch noch Aussagen über deren *Relevanz* machen können.

Mit meiner Forschung will ich einen ersten Schritt machen, um diese Lücke zu schliessen. In dieser Studie konzentriere ich mich ausschliesslich auf das Motiv „Spass am Programmieren“. Es geht mir also darum zu erklären, welcher Anteil des Engagements der Open-Source-Programmierer durch dieses Motiv erklärt werden kann.

[1] Siehe z.B. die Studien von Lerner und Tirole (2001), Hars und Ou (2001), Osterloh, Rota und Kuster (2002a), Franck und Jungwirth (2002b), Lakhani und Wolf (2003), Hertel, Niedner und Herrmann (2003).

1.1 Forschungsfragen

Die zentrale Forschungsfrage meiner Dissertation lautet:

Forschungsfrage 1 *Welche Bedeutung hat Spass als Motivation für einen Open-Source-Entwickler?*

Oder präziser formuliert:

Wie gross ist der Anteil des Engagements eines Open-Source-Entwicklers, welcher durch ein Modell erklärt werden kann, in welchem Spass und Freizeit als unabhängige Variablen vorkommen?

Zusatzfragen

Open Source ist in erster Linie ein Lizenzierungs-Modell für Software. Aus diesem speziellen Lizenzierungs-Modell ergibt sich naheliegenderweise ein eigenes Software-Entwicklungsmodell (siehe Kapitel 3.3). Ein Aspekt dieses Entwicklungsmodells ist, dass Open-Source-Software traditionell in der Freizeit und unentgeltlich entwickelt wird. Das Lizenzierungs-Modell für Open-Source-Software schliesst aber nicht aus, dass solche Software auch unter kommerziellen Bedingungen entwickelt wird. Es stellt sich deshalb folgende Frage:

Forschungsfrage 2 *Wie gross ist der Anteil der Open-Source-Software, der in der Freizeit entwickelt wird? Wie viel Open-Source-Software wird in einem kommerziellen Umfeld und gegen Lohn entwickelt?*

Hinter meiner Forschungsfrage steht die Überlegung, dass sich das Phänomen „Open Source“, d.h. die kostenlose Bereitstellung von qualitativ guter Software, dadurch erklären lässt, dass die Tätigkeit „Programmieren“ Spass macht. Open-Source-Entwickler programmieren in ihrer Freizeit, so die Idee, weil sie den dabei erlebten Spass konsumieren und das Produkt „Open-Source-Software“ ein Nebenprodukt dieser Tätigkeit ist. Dieses Argument tönt stichhaltig, reicht aber noch nicht aus, um die Existenz von Open-Source-Software zu begründen. Wenn Programmieren Spass macht, so erklärt dies, dass es Personen gibt, die Software entwickeln. Zu berücksichtigen ist aber die Tatsache, dass man mit Software Geld verdienen kann. Wenn nun aber gilt, dass Programmieren Spass macht und man gleichzeitig damit Geld verdienen kann, was zweifellos einen zusätzlichen Nutzen darstellt, dann erwarten wir eigentlich, dass überhaupt keine Person in ihrer Freizeit Software entwickelt, weil eben Spass haben *und* Geld verdienen *mutatis mutandis* besser ist als bloss Spass haben.

Die Möglichkeit, dass Software-Entwickler auch in ihrer Freizeit programmieren, wird dann verständlich, wenn wir zusätzlich postulieren, dass das Open-Source-Entwicklungsmodell den Programmierern bessere Möglichkeiten bietet, Spass zu erleben, als das kommerzielle Software-Entwicklungsmodell. Diese Hypothese will ich verifizieren, indem ich die Open-Source-Entwickler, die sich in ihrer Freizeit an Open-Source-Projekten beteiligen, mit Programmierern vergleiche, die unter kommerziellen Bedingungen Software entwickeln (Open Source oder proprietär), und die folgende Frage stelle:

Forschungsfrage 3 *Erleben Software-Entwickler in Open-Source-Projekten mehr Spass als Programmierer, die unter kommerziellen Bedingungen Software entwickeln?*

Falls diese Frage verifiziert werden kann, weil ein Vergleich der Open-Source-Programmierer und der kommerziellen Software-Entwickler tatsächlich zu signifikanten Unterschieden bezüglich des Erlebens von Spass führt, so stellt sich die Frage nach der Ursache dieser Differenz:

Forschungsfrage 4 *Sind die Merkmale eines Open-Source-Projekts, an welchem sich Entwickler in ihrer Freizeit beteiligen (fehlende Abgabetermine, klare Projektvision, optimale Herausforderung, keine monetären Anreize für Projekt-Mitarbeiter, keine formale Autorität des Projekt-Leaders), die Ursache, dass Entwickler in Open-Source-Projekten mehr Spass empfinden als Programmierer in kommerziellen Software-Projekten?*

1.2 Zur Relevanz der Forschungsfragen

Warum ist die Beantwortung dieser Fragen relevant? Das Beispiel von Microsoft zeigt, dass mit Software sehr viel Geld verdient werden kann. Bezüglich der Qualität der von Microsoft angebotenen Software gehen die Meinungen in der Fachwelt auseinander. Umgekehrt ist auch die Qualität von Open-Source-Software nicht in jedem Fall überzeugend. Unbestreitbar ist aber, dass es qualitativ hervorragende Open-Source-Software gibt. Die Beispiele von Linux (Betriebssystem), Apache (Webserver), von OpenOffice.org (Office-Suite) und Eclipse (Tools-Platform) belegen die Existenz von Software, die vom Funktionsumfang und von der Qualität her mit dem entsprechenden Produkt von Microsoft gleichgezogen hat und *kostenlos* verfügbar ist.

Warum gibt es Software-Entwickler, die ohne Entgelt Zeit investieren für die Herstellung eines Produkts, das in der Folge kostenlos zur Verfügung gestellt wird, das aber problemlos verkauft werden und damit den beteiligten Personen einen Verdienst ermöglichen könnte? Ist das Bild des rational seinen Nutzen maximierenden Homo oeconomicus falsch? Oder bietet das Open-Source-Entwicklungsmodell den beteiligten Programmierern einen Nutzen, der sie für die Opportunitätskosten, den nicht realisierten Verdienst entschädigt?

Mit der Beantwortung meiner Forschungsfragen geht es im Allgemeinen darum zu entscheiden, ob das Modell des Homo oeconomicus auch im Bereich „Open Source“ angewendet werden kann oder ob dieser Bereich einer klassischen ökonomischen Betrachtungsweise nicht zugänglich ist.

In der wissenschaftlichen Analyse des Open-Source-Phänomens sind auf Grund von theoretischen Überlegungen (z.B. Lerner und Tirole (2001)) wie auch von qualitativen Untersuchungen (z.B. Ghosh, Glott, Krieger und Robles (2002)) eine Reihe von Motivationen genannt worden, die auch auf dem Hintergrund des Homo-oeconomicus-Modells das Verhalten der Akteure im Open-Source-Bereich verständlich machen können. Bis jetzt fehlen aber weitgehend solche Untersuchungen, welche die Bedeutung dieser verschiedenen Motivationsfaktoren quantifizie-

ren. Im Speziellen will ich mit meiner quantitativen Untersuchung diesen Mangel zumindest für den Faktor „Spass“ korrigieren.

Es gibt mehrere Gründe, warum ich mich gerade und ausschliesslich auf das Motiv „Spass am Programmieren“ konzentriere. Einerseits taucht dieses Motiv in den bisherigen empirischen und qualitativen Studien zum Open-Source-Phänomen regelmässig auf (siehe beispielsweise Hars und Ou (2001), Ghosh u. a. (2002) oder Lakhani und Wolf (2003)). Andererseits gilt es als selbstverständlich, weshalb bisher wenig Forschungsaufwand eingesetzt worden ist, um dieses Motiv vertieft zu untersuchen.

Weiter lässt die Beschäftigung mit dem Motiv „Spass“ interessante betriebswirtschaftliche Implikationen zu. Wenn sich durch meine Forschung herausstellen sollte, dass die Freude am Programmieren von wesentlicher Bedeutung ist für das freiwillige und unbezahlte Engagement von Open-Source-Entwicklern, so stellt sich zwangsläufig die Frage, wie viel Spass denn Software-Entwickler im kommerziellen Umfeld haben. Ist es möglich, dass Software-Entwickler, die unter kommerziellen Bedingungen programmieren, weniger Freude an dieser Tätigkeit erleben als die Open-Source-Programmierer? Und wenn es mir gelingt, tatsächlich einen Unterschied aufzuzeigen, so lässt sich weiter fragen: Was ist die Ursache, dass die Tätigkeit „Software entwickeln“ mehr Spass macht, wenn sie im Kontext von Open-Source-Projekten ausgeübt wird? Wenn ich es fertig bringe, solche Ursachen zu identifizieren, so führt dies weiter zu der Frage: Sind diese Ursachen im Open-Source-Entwicklungsmodell begründet und damit diesem spezifischen Entwicklungsmodell inhärent, oder ist es prinzipiell möglich, das Entwickeln von Software so zu gestalten, dass Programmierer auch unter kommerziellen Bedingungen gleichermassen Spass empfinden?

In meiner Dissertation geht es also um das „Warum“ bei Open-Source-Software. Mich interessiert die Motivation der individuellen Beitragsleistenden.

Damit grenze ich mich von den Untersuchungen im Open-Source-Bereich ab, die das „Wie“ von solchen Software-Projekten erforschen. Die Frage, wie die Teamproduktion „Open-Source-Software“ zustande kommt, wie die vielen grossen und kleinen Beiträge wirksam koordiniert werden können, so dass ein einheitliches und funktionsfähiges Produkt entsteht, ist nicht Gegenstand dieser Untersuchung.

Ebenfalls nicht Gegenstand meiner Forschung ist die Frage nach den Erfolgsbedingungen von Open-Source-Projekten. Unter welchen Bedingungen ein Projekt eine Community und Akzeptanz auf dem Software-Markt findet, oder aber ein gebrauchsfähiges Stadium nie erreicht, werde ich in dieser Arbeit nicht untersuchen.

1.3 Gliederung

Meine Dissertation ist in zwei Teile gegliedert. Im ersten, theoretischen Teil bereite ich den wissenschaftlichen Boden vor, auf welchem sich meine Forschung abspielt. In Kapitel 2 entwerfe ich das Modell, mit welchem ich meine Forschungsfragen wissenschaftlich behandeln will, und beschreibe die Operationalisierung der darin enthaltenen Variablen. Im darauf folgenden Kapitel erörtere ich die verschiedenen Facetten des Open-Source-Phänomens, stelle das für die Open-Source-Programmierer relevante Anreizsystem dar und bette meine Forschungsfragen ein. Die bei-

den daran anschliessenden Kapitel widme ich methodischen Fragen. Spass am Programmieren operationalisiere ich mit dem Flow-Konzept, welches im vierten Kapitel erläutert wird, während es im fünften Kapitel um die Fragebogenmethode im Allgemeinen und Online-Umfragen im Speziellen geht. Im letzten Kapitel des theoretischen Teils geht es um das Design meiner Umfrage, mit welcher ich die zur Beantwortung der Forschungsfragen benötigten Daten erhoben habe.

Im zweiten Teil meiner Dissertation werte ich meine Forschungsdaten systematisch aus. Diese Daten sind das Resultat zweier Umfragen unter Software-Entwicklern, wobei die einen an Open-Source-Projekten beteiligt sind, während die zweiten in Schweizer Firmen unter kommerziellen Bedingungen arbeiten. Entsprechend fängt die Präsentation der Forschungsresultate im siebten Kapitel mit der Analyse der Daten der Open-Source-Umfrage an, während ich im achten Kapitel die Daten aus der kommerziellen Umfrage auswerte. Um einen allgemeinen Eindruck über die Population der Open-Source-Entwickler zu bekommen, startet die Analyse der Open-Source-Umfrage mit deskriptiven Auswertungen der Daten. Im anschliessenden Unterkapitel evaluiere ich die Daten, die ich zum Engagement der Open-Source-Entwickler erhoben habe. In Unterkapitel 7.3 studiere ich die Daten zum Flow-Empfinden der Beitragsleister. Mit einer Faktorenanalyse kann ich die in der Umfrage erhobenen Daten auf zugrunde liegende Faktoren zurückführen. Im nächsten Unterkapitel verwende ich die erhobenen Daten für eine Typisierung der Open-Source-Programmierer. Damit habe ich meine Daten so weit konsolidiert, dass ich sie im abschliessenden Unterkapitel in mein Modell einfügen und damit die Frage nach der Bedeutung des Spasses am Programmieren für das Engagement der Open-Source-Entwickler beantworten kann.

In Unterkapitel 8.1 beginne ich die Analyse der Daten aus der kommerziellen Umfrage. Dieses Kapitel starte ich wieder mit deskriptiven Auswertungen. Das darauf folgende Unterkapitel enthält die Faktorenanalyse der Daten zum Flow-Empfinden der Software-Entwickler, diesmal solcher aus dem kommerziellen Bereich. In Unterkapitel 8.3 untersuche ich die Bedingungen der kommerziellen Software-Produktion, indem ich die Daten der kommerziellen Umfrage in unterschiedlichen Kombinationen korreliere. Im letzten Unterkapitel der Auswertungen der Daten aus der kommerziellen Umfrage versuche ich eine Typisierung der Software-Entwickler, die unter kommerziellen Bedingungen arbeiten.

Die Frage nach der Existenz von Unterschieden zwischen dem Open-Source- und dem kommerziellen Entwicklungsmodell und deren Ursachen behandle ich in Kapitel 9, in welchem ich die beiden Datensätze direkt vergleiche. Meine Forschungsarbeit beende ich in Kapitel 10 mit den Schlussfolgerungen aus meiner Untersuchung.

Teil I

Theorie

Kapitel 2

Theoretisches Konzept

2.1 Modell

Um die Frage nach der Bedeutung von Spass für die Erklärung des freiwilligen und unbezahlten Engagements für Open Source zu erklären, brauche ich ein Modell, welches Spass als unabhängige Variable mit Engagement als abhängige Variable verknüpft. Ein solches Modell hilft mir nur weiter, wenn ich nicht direkt nach der Bedeutung von Spass frage, sondern untersuche, warum die einen Entwickler mehr Engagement zeigen als die anderen. Ich analysiere also die Varianz des Engagements für Open Source. Mit einer statistischen Regressionsanalyse kann ich in der Folge ermitteln, welcher Anteil dieser Varianz mit meinem Modell erklärt werden kann. Der Erklärungsgehalt meines Modells bei der Frage, warum die einen Entwickler ein grosses Engagement zeigen, ist aber nichts anderes als der gesuchte quantitative Anhaltspunkt für die Bedeutung von Spass.

Das Modell, das ich in diesem Fall suche, kann auch als Produktionsfunktion aufgefasst werden, mit welcher ein funktionaler Zusammenhang zwischen Input und Output hergestellt wird. In meinem Fall handelt es sich beim Output um „Engagement für Open Source“, während einer der Inputfaktoren aus „Spass am Programmieren“ besteht. Wenn das Phänomen, das erklärt werden soll, das freiwillige und unbezahlte Engagement für Open Source ist, dann weist dies darauf hin, dass es sich dabei um ein Engagement handelt, das in der Freizeit geleistet wird. Auf Grund dieser Überlegung ist klar, dass auch „Freizeit“ als weiterer Inputfaktor Teil der gesuchten Produktionsfunktion sein muss. Weiter unterstelle ich einen abnehmenden Grenzeffekt: Eine zusätzliche Einheit in einem Inputfaktor führt nicht zu einem linearen Anstieg im Output, sondern bloss zu einer unterproportionalen Steigerung des Engagements.

Eine Produktionsfunktion, welche diesen Anforderungen genügt, hat folgende quadratische Form:

$$E = c + a_1 * S - a_2 * S^2 + b_1 * Z - b_2 * Z^2 \quad (2.1)$$

wo E: freiwilliges, unentgeltliches Engagement
S: Spass
Z: Freizeit
$a_1, a_2, b_1, b_2 > 0$

In Anhang B zeige ich, wie eine solche Produktionsfunktion theoretisch hergeleitet werden kann.

Eine alternative Produktionsfunktion, welche ebenfalls einen abnehmenden Grenzeffekt zeigt, kann mit einem logarithmischen Modell erreicht werden:

$$E = c + a * \ln S + b * \ln Z \quad (2.2)$$

wo E: freiwilliges, unentgeltliches Engagement
S: Spass
Z: Freizeit
$a, b > 0$

Es stellt sich allerdings die Frage, ob die Annahme eines sinkenden Grenzeffekts im Falle von Spass wirklich angebracht ist. Im Zusammenhang mit Programmieren, speziell im Kontext von Open Source, kann man sich durchaus vorstellen, dass das Produkt, der Programm-Code, das mehr oder weniger intendierte Nebenprodukt einer Tätigkeit ist, welche eminent Spass macht. Wenn es aber in erster Linie um den Spass geht, welcher mit der Tätigkeit konsumiert werden kann, dann ist die Annahme eines abnehmenden Grenzeffekts nicht angebracht.

Umgekehrt spricht aber nichts dagegen, ein quadratisches Modell zu testen. Ist die Annahme eines sinkenden Grenzeffekts falsch, so wird dies in der Regressionsanalyse sichtbar. Im Falle einer Regressionsanalyse lautet die Nullhypothese, dass eine Änderung in einer unabhängigen Variablen keine Auswirkung auf die abhängige Variable hat. Ergibt die Regressionsanalyse für einen Korrelationskoeffizienten aber einen signifikanten Wert, so kann die Nullhypothese verworfen werden. In einem solchen Fall kann von einem signifikanten Effekt der Variablen ausgegangen werden. Entsprechend müsste ein sinkender Grenzeffekt in einem signifikanten Wert der Korrelationskoeffizienten der quadratischen Terme zum Ausdruck kommen.

2.2 Operationalisierung

Es stellt sich nun die Frage, wie die Variablen in diesem Modell, Engagement, Spass und Freizeit operationalisiert werden können.

Das Engagement in Open-Source-Projekten habe ich auf drei unterschiedliche Arten operationalisiert. Einerseits frage ich die Probanden nach ihrer Bereitschaft, sich auch in Zukunft an Open-Source-Projekten zu beteiligen. Eine solche Fragestellung hat sich in der empirischen Untersuchung von Hertel u. a. (2003) bewährt. Als zweites frage ich, wie viel Freizeit die Open-Source-Entwickler in

diese Tätigkeit investieren. Als dritte Möglichkeit versuche ich, das Engagement über die Anzahl der geleisteten Patches (Software-Bestandteile) zu ermitteln.

Für die Ermittlung von Spass stütze ich mich auf die Arbeiten, die im Zusammenhang mit dem Flow-Konzept von Csikszentmihalyi (in Csikszentmihalyi, 1975) entwickelt worden sind. Flow ist eine spezielle Form von Spass. Das Flow-Konzept scheint mir im Zusammenhang mit Tätigkeiten am Computer allgemein und für das Programmieren speziell ein ausgesprochen fruchtbares Konzept zu sein. Seit Csikszentmihalyi das Flow-Konzept erstmals präsentiert hat, sind eine Fülle von Studien erschienen, in denen Flow gemessen worden ist (z.B. Rheinberg, Vollmeyer und Engeser (2002), Montgomery, Sharafi und Hedman (2004), Chen (2002), Novak, Hoffman und Yung (1998)). Mein Teil der Untersuchung, der sich mit Flow beschäftigt, ist stark geprägt vom Fragebogen, den Remy (2002) für die Ermittlung von Flow-Erleben bei Tätigkeiten am Computer entwickelt hat.

Die Freizeit der Probanden kann ich auf zwei Arten berechnen. Einerseits kann ich mit Hilfe einer Ergänzungsfrage diesen Wert bestimmen, wenn ich die Probanden nach ihrem Open-Source-Engagement in der Freizeit befrage. Andererseits greife ich auf eine Arbeit von Bittman und Goodin (1998) zurück. Diese Autoren sind der Frage nachgegangen, ob mit einem einfachen Modell die gesamte wöchentlich geleistete Arbeitszeit (d.h. bezahlt und unbezahlt) berechnet werden kann. Sie entwickelten ein einfaches Modell, basierend auf der wöchentlichen bezahlten Arbeitszeit, der Existenz von Kindern sowie der Präsenz von weiteren Erwachsenen im Haushalt. Um die Qualität dieses Modells zu überprüfen, verglichen sie die mit dem Modell berechneten Daten mit den Werten einer Umfrage, in welcher die Probanden direkt nach dem gesamten Umfang ihrer wöchentlichen Arbeitszeit gefragt worden sind. Wie die Analyse ergab, konnten mit dem Modell für Männer erstaunliche 91% der gesamten wöchentlichen Arbeitszeit erklärt werden, für Frauen immerhin noch 59%. Wenn ich mit Hilfe dieses Modells die gesamte wöchentliche Arbeitszeit einer Person kenne, kann ich die ihr zur Verfügung stehende Freizeit berechnen als Differenz zwischen der gesamten Zeit pro Woche und der mit Hilfe dieses Modells berechneten wöchentlichen Arbeitszeit.

2.3 Der Stand der Forschung

Was trägt meine Untersuchung zum Stand der Forschung auf dem Gebiet „Open Source“ bei? Eine Voraussetzung dafür, dass die Resultate meiner Untersuchung nicht entweder irrelevant sind oder aber längst bekanntes Wissen reproduzieren, ist, dass ich mit dem aktuellen Forschungsstand auf dem Gebiet vertraut bin. Den Stand der Forschung im Bereich „Open Source“ reflektiere ich in Kapitel 3, in Tabelle 3.3 stelle ich die Eckwerte der empirischen Arbeiten auf diesem Gebiet zusammen.

Ein aktueller Wissensstand ist nicht nur bezüglich des Forschungsgegenstands notwendig. Auch bei den eingesetzten Methoden und verwendeten Konzepten ist es angebracht, auf der Höhe des Wissens zu sein.

2.3.1 Flow

Das Flow-Konzept führe ich in Kapitel 4 aus. Der aktuelle Forschungsstand wird weiterhin von Csikszentmihalyi geprägt (Csikszentmihalyi, 2004). Intensiv mit diesem Thema beschäftigt sind auch die Forschungsgruppen von Rheinberg (z.B. Rheinberg u. a. (2002)) und Schallberger (siehe Schallberger und Pfister (2003)). Wertvoll im Zusammenhang von Flow mit dem Verhalten im Internet sind die Arbeiten von Novak und Hoffman (z.B. Novak und Hoffman (1999)) und Chen (2002). Als sehr nützlich für meine Arbeit hat sich die Untersuchung von Remy (2002) und ihre darin gemachten Überlegungen erwiesen.

2.3.2 Online-Umfragen

Der technische Fortschritt, welcher durch den Einsatz von Computern möglich wird, zeigt sich auch bei den Umfragetechniken. Computer-Unterstützung und das Internet haben bei wissenschaftlichen Umfragen Einzug gehalten.

Für die Datenerhebung der FASD-Studie habe ich einen Online-Fragebogen verwendet. Die Methodologie der Online-Umfrage führe ich in Kapitel 5 aus. Beim Entwurf und der Gestaltung der Online-Umfrage habe ich mich von Bandilla und Hauptmanns (1999), Bosnjak und Batinic (1999) und speziell Batinic (2001) leiten lassen.

Kapitel 3

Das Phänomen „Open Source“

In diesem Kapitel gebe ich einen Abriss über meinen Forschungsgegenstand „quelloffene Software“. Im ersten Unterkapitel gehe ich auf die lizenztechnischen Eigenheiten ein, welche Open-Source-Software als öffentliches Gut charakterisieren. Aus diesen speziellen Eigentumsrechten am Code ergibt sich ein charakteristisches Open-Source-Entwicklungsmodell, das ich im darauf folgenden Unterkapitel beschreibe. Das Verständnis dieses Entwicklungsmodells erlaubt es, den Fokus auf den Benutzer-Programmierer als treibende Kraft beim Open-Source-Phänomen zu verschieben. Die Untersuchung der Motivation dieser Benutzer-Programmierer kann in der Folge erklären, warum im Falle von Open-Source-Software das Problem der Unterversorgung, eine sonst übliche Schwierigkeit bei öffentlichen Gütern, gelöst werden kann. Eines der Motive, die in diesem Zusammenhang immer wieder genannt werden, ist die Freude am Programmieren. Was Spass beim Programmieren im Kontext von Open Source genau heisst, darauf gehe ich in einem eigenen Unterkapitel ein, bevor ich im abschliessenden Teil den Stand der Forschung erläutere.

3.1 Open-Source-Software als öffentliches Gut

Open-Source-Software bedeutet im Wesentlichen, dass der Quellcode der Software frei verfügbar ist. Im Speziellen bedeutet Open Source ein Lizenzierungsmodell, welches den Kriterien der *Open Source Initiative* entspricht (vgl. Open Source Initiative (1999)).[1]

Ein wesentliches Merkmal einer solchen Lizenz ist, dass jedem Nutzer ausdrücklich das Recht zugestanden wird, den Quellcode zu kompilieren und als lauffähige Applikation auf seinem Rechner zu installieren (siehe Open Source Initiative, 1999, Def. 2 und 6).

Aus dieser Definition von Open Source folgt weiter, dass mit einem solchen Software-Lizenzierungsmodell öffentliche Güter geschaffen werden. Von der Nutzung von Open-Source-Software wird keine Person ausgeschlossen (Nicht-Ausschliessbarkeit, explizit in Def. 5 und 7 der Open-Source-Definitionen). Die Nutzung solcher Software durch die eine Person beeinträchtigt in keiner Weise die

[1] Die Open-Source-Definitionen sind in Anhang A wiedergegeben.

Möglichkeit anderer Personen diese Software zu nutzen (Nicht-Rivalität des Konsums).

3.2 Urheberrecht und Lizenz

Die Open-Source-Definitionen gemäss *Open Source Initiative* lassen ganz unterschiedliche Realisierungen zu. Im Wesentlichen unterscheiden sich die verschiedenen Varianten von Open-Source-Lizenzen im Hinblick darauf, welche Regeln sie für abgeleitete Werke definieren. Als abgeleitete Werke gelten Programme, die in irgendeiner Form den ursprünglichen, unter einer Open-Source-Lizenz veröffentlichten Code enthalten. Konkret heisst dies, dass der ursprüngliche Code (oder Teile davon) in den Quellcode der neuen Applikation kopiert wird oder dass Module des ursprünglichen Codes so mit der neuen Applikation verlinkt werden, dass diese nur in Maschinencode übersetzt werden kann, wenn die ursprüngliche Applikation auf dem System installiert ist.

Eine strikte, so genannte virale Lizenz wie die *GNU General Public Licence (GPL)* verlangt, dass abgeleitete Werke ebenfalls unter einer GPL freigegeben werden müssen, wenn der ursprüngliche Code der GPL unterliegt. Ein abgeleitetes Werk bindet häufig Codes aus verschiedenen Quellen ein. Welche Bedingungen gelten nun für ein abgeleitetes Werk *C*, welches Codes sowohl aus der Quelle *A* wie auch *B* einbindet, und der Code *B* der GPL unterstellt worden ist? In einem solchen Fall kann *C* nur (unter GPL) freigegeben werden, wenn der Code *A* auch der GPL (oder einer kompatiblen Lizenz) unterliegt.

Die GPL führt also zu bedeutenden Einschränkungen bei der Kombination von Fremdcodes zu einem kombinierten abgeleiteten Werk. Im Gegensatz zur GPL schränken liberalere Open-Source-Lizenzen wie z.B. die *Berkley Software Distribution (BSD)* oder die *Apache Licence* solche Kombinationen in keiner Weise ein. Bei der Lizenzierung mit liberalen Varianten ist es möglich, dass selbst proprietäre Applikationen Open-Source-Codes einbinden.

Die Wahl der geeigneten Lizenz für Open-Source-Software ist in verschiedener Hinsicht relevant. Entscheidet sich ein Open-Source-Projekt, den Code mit der GPL zu versehen, so schliesst es die Möglichkeit aus, dass kommerzielle Projekte diesen Code in ihren proprietären Applikationen einbinden können. Dies kann die Weiterverbreitung des Codes unter Umständen behindern. Bedeutsam ist dieser Sachverhalt, wenn es sich bei diesem GPL-Code nicht um eine weitgehend abgeschlossene Computeranwendung handelt, sondern um einen Basisdienst oder ein Framework.

Die offene Tools-Platform *Eclipse* ist ein Beispiel für ein solches Framework. Eclipse biete eine grosse Menge von Erweiterungspunkten an, die es anderen Anbietern erlaubt, eigene Plug-ins zu entwickeln und auf der Basis von Eclipse vollständige Applikationen zu erzeugen. Eclipse hat auf diese Weise innert kurzer Zeit eine sehr hohe Akzeptanz gefunden, und dies sowohl unter Open-Source-Entwicklern wie auch unter kommerziellen Anbietern. Der Grund ist zweifellos, dass Eclipse mit der *Eclipse Public License (EPL)* sehr liberal lizenziert wurde. Die EPL erlaubt die Entwicklung von Plug-ins nicht nur unter Open-Source-, sondern auch unter proprietären Lizenzen. Mit dieser Vorgehensweise hat das Eclipse-

Konsortium[2], das von massgebenden Software-Unternehmen getragen wird, eine gute Chance, mit Eclipse einen neuen de-facto-Standard im Software-Bereich zu setzen.

Eine interessante Lizenzierungs-Variante ist für die Open-Source-Datenbank *MySQL*[3] gewählt worden. Eine Datenbank ist ein klassischer Basisdienst. Viele Applikationen manipulieren Daten, die in irgendeiner Weise persistent gehalten werden. Eine Datenbank ist eine geeignete und weit verbreitete Möglichkeit für die Speicherung von Daten. MySQL wird unter der GPL angeboten. Diese Lizenzierungsart, zusammen mit der Leistungsfähigkeit der Datenbank, hat zweifellos zur grossen Verbreitung des Produkts beigetragen. Der Einstieg in die Behandlung von MySQL-Datenbanken ist leicht zu erlernen, entsprechend vertraut sind viele Software-Entwickler im Umgang mit dieser Datenbank. Für kommerzielle Anbieter, welche diesen Basisdienst in ihr Produkt integrieren möchten, ist aber eine GPL ungeeignet. Für diese Kundschaft verkaufen nun die Eigentümer die MySQL-Datenbank unter massgeschneiderten Lizenzen. Diese *dual-licensing*-Strategie ermöglicht es den Eigentümern, sowohl mit der Open-Source-Version von MySQL eine Entwickler-Gemeinschaft aufzubauen und mit der weiten Verbreitung des Produkt eine hohe Akzeptanz zu schaffen, zugleich aber mit der Möglichkeit der kommerziellen Lizenzierung mit ihrem Produkt ein Einkommen zu erzielen.

Es gibt aber auch Argumente, die für einen Einsatz der GPL sprechen. Wie ich in Abschnitt 3.5.5 ausführe, kann das Engagement im Open-Source-Bereich als Code-Spende altruistisch motivierter Programmierer verstanden werden. Gemäss Franck und Jungwirth (2003) ist die Argumentation von Hansmann (Hansmann, 1980) gut geeignet, dieses Verhalten zu verstehen: Rationale Spender wollen sicher sein, dass ihre Gabe dem Unternehmensziel entsprechend verwendet wird und nicht privat angeeignet werden kann. Das institutionelle Arrangement, welches diesen Spenderwunsch einlösen kann, ist die Form des Non-Profit-Unternehmens. In analoger Weise will ein altruistisch motivierter Open-Source-Entwickler sicherstellen, dass seine Code-Spende nicht privat, d.h. proprietär angeeignet wird. Das institutionelle Arrangement in Form der geeigneten Open-Source-Lizenz wird am besten von der GPL erfüllt. Nur eine virale Lizenz verhindert, dass eine Code-Spende in einem abgeleiteten Werk in proprietärer Form eingesetzt werden kann. Entsprechend dieser Argumentation sollten Open-Source-Projekte, die unter der GPL laufen, für altruistisch motivierte Entwickler besonders attraktiv sein.

Diese Überlegungen zeigen, wie die Wahl der Open-Source-Lizenz das Potential des Projekts und dessen spätere Entwicklung auf die eine oder andere Weise beeinflusst.

Grundsätzlich besitzt jede Person, die intellektuelles Eigentum herstellt, das Urheberrecht auf diesem Eigentum. Mit dem „Besitz“ eines intellektuellen Eigentums hat dessen Eigentümer auch die Möglichkeit, mit einer geeigneten Lizenzbestimmung die Nutzung dieses intellektuellen Eigentums zu definieren.

Wird dieses Recht aber von allen Beitragsleistenden wahrgenommen, würde also jeder Beitrag autonom lizenziert, so bestände die Gefahr, dass das Open-

[2] siehe www.eclipse.org/
[3] siehe www.mysql.org/

Source-Projekt in disparate Teile zerfallen und in seiner Weiterentwicklung blockiert würde. In der Regel entscheidet der Projekteigentümer über die Lizenz, während die Beitragsleister diese Entscheidung zumindest implizit mittragen oder sogar explizit auf das Recht zur Lizenzierung ihrer Beiträge verzichten.[4]

Für ein exaktes Verständnis der Motivationen von Open-Source-Entwicklern ist es wichtig zu unterscheiden zwischen Entwicklern, welche das Recht haben, über die Lizenz der Software zu bestimmen, und solchen, die dieses Recht nicht haben. Im Folgenden bezeichne ich die erste Gruppe als die Projekteigentümer oder Projektleiter, die zweite als Programmierer. Eine solche Differenzierung ist nützlich, weil Projekteigentümer und Programmierer einen in gewissen Bereichen unterschiedlichen Handlungsspielraum haben, entsprechend können ihre Handlungen von unterschiedlichen Motiven gesteuert sein.

3.3 Das Open-Source-Entwicklungsmodell

Die oben ausgeführten Merkmale der Open-Source-Lizenzen führen zu einem spezifischen Software-Entwicklungsmodell[5] (vgl. Dafermos (2001), Sieckmann (2001), Weber (2000)) mit den in Tabelle 3.1 aufgeführten Komponenten.

- Die Benutzer können sich den Quellcode anschauen und darin Fehler korrigieren und Veränderungen vornehmen. Die Benutzer tragen so direkt zur Entwicklung der Applikation bei. Um das Open-Source-Projekt formiert sich eine *personell offene und geographisch verteilte Entwicklergemeinschaft*.
- Die *Entwickler* sind meistens auch *Benutzer der Applikation*, umgekehrt tragen viele Benutzer zur Weiterentwicklung der Software bei.
- Der Grossteil der Beiträge am Open-Source-Produkt kommt auf *freiwilliger Basis* zustande.
- Da sich die Entwicklergemeinschaft auf freiwilliger Basis konstituiert, hat der *Projekteigentümer keine formale Autorität* innerhalb des Open-Source-Projekts. Seine Autorität beruht auf fachlicher und sozialer Kompetenz.
- Die Grenzen der Entwicklergemeinschaft eines Open-Source-Projekts sind offen und durchlässig. Jeder Benutzer ist ein potentieller Entwickler. Während aber die reinen Benutzer stabil laufende, möglichst fehlerfreie Software wünschen, wollen die Entwickler-Benutzer die Software lieber mit den neusten Features und sind dabei bereit, Programmfehler in Kauf zu nehmen. Aus diesem Grund wird Open-Source-Software oft in zwei (oder mehr) Versionen angeboten. Die Entwickler-Version integriert die jeweils neusten Entwicklungen und lädt die Benutzer zur Fehlererkennung und -behebung ein.

[4] Die *Apache Software Foundation* verlangt von allen Projekten, die unter ihrem Dach entstehen, die Übertragung des Copyrights auf allen Code-Beiträgen. Die *Python Software Foundation (PSF)* verlangt von ihren Beitragsleistenden das Recht auf die Lizenzierung. Die PSF erhält so die Möglichkeit, autonom über eine allfällige Re-Lizenzierung des Codes zu entscheiden.

[5] Raymond hat in (Raymond, 2000b) den Begriff *Bazaar-Style* für dieses Entwicklungsmodell geprägt.

Diese Version der Software wird schon in einem *frühen Reifegrad* und mit *grosser Releasehäufigkeit* freigegeben. Sobald die Software einen akzeptierten Reifegrad erreicht hat, wird sie als Version für die reinen Benutzer freigegeben. Die technologische Entwicklung findet in der Folge ausschliesslich auf einer neuen Releaselinie für die Entwickler statt, während an der Benutzerversion nur noch Fehlerkorrekturen vorgenommen werden.[6]

- Der Status der Beitragsleistenden wird durch *meritökonomische Mechanismen* gesteuert. Die Reputation eines Beitragsleistenden beruht auf der Wertschätzung, die Peers der erbrachten Leistung entgegenbringen.

Tabelle 3.1: Komponenten des Open-Source-Entwicklungsmodells

3.4 Die Sicht der Benutzer-Programmierer

Mit Software kann man viel Geld verdienen. Sowohl der Einsatz wie auch die Erzeugung von Software machen erhebliche Gewinne möglich. Dazu kommt, dass Computer und Software-gesteuerte Apparate den Alltag, vor allem den geschäftlichen, durchdringen. Diese Erfahrungen machen es nahe liegend, Phänomene im Zusammenhang mit Software in erster Linie durch die wirtschaftliche Brille zu betrachten.

In einer solchen wirtschaftlich geprägten Sichtweise erscheint das Open-Source-Phänomen offensichtlich als Paradox: Warum engagieren sich Leute, um ein qualitativ hoch stehendes Produkt zu erzeugen, wenn dieses in der Folge frei zur Verfügung gestellt wird? Warum verschenkt die eine Person ein wertvolles Gut, wo doch die andere damit viel Geld verdient?

Eine Möglichkeit, dieses Paradox aufzulösen, ergibt sich aus der Verschiebung des Blickwinkels. Statt, wie in der ökonomisch geprägten Sichtweise nahe liegend, die Software-Firma als den relevanten Akteur zu betrachten, werden die Software-Benutzer-Programmierer ins Zentrum der Untersuchung gerückt. Im Zentrum der Analyse des Open-Source-Phänomens stehen dann die „Prosumer“ (siehe Toffler, 1981), d.h. Benutzer, welche die Software an ihre Bedürfnisse anpassen und weiter entwickeln. Wie von Hippel und von Krogh (in von Hippel und von Krogh, 2002) gezeigt haben, ist mit einem solchen Perspektivenwechsel ein wesentlicher Erkenntnisgewinn möglich.

Wesentlich dabei ist, dass das Bild von rational handelnden Akteuren nicht verworfen werden muss. Prosumer handeln nicht weniger vernünftig als Software-Firmen, wenn sie in die Herstellung von Software investieren. Für beide Arten von Akteuren gilt, dass Software dann freigegeben wird, wenn der Nutzen einer Offenlegung des Quellcodes grösser ist als die Kosten einer solchen Handlung. Nur präsentiert sich das Umfeld und damit auch die Rechnung für die Software-Firmen anders als für die Prosumer. Software-Firmen befinden sich in einem ausgeprägten

[6] Vgl. Raymond (2000b, S. 7): „Release early, release often“.

Konkurrenzverhältnis zueinander. Für sie ist eine Freigabe des Quellcodes mit hohen Opportunitätskosten verbunden: Mit einer Offenlegung des Quellcodes geben sie ein Geschäftsgeheimnis preis und verspielen dadurch einen Wettbewerbsvorteil.

Für die Software-Benutzer-Programmierer sieht die Lage ganz anders aus. Das Verhältnis der Prosumer untereinander ist durch bloss geringe Rivalitäts-Bedingungen bestimmt. Zusätzlich gilt, dass dank dem Internet auch die direkten Kosten gering sind: die Verbreitung des Quellcodes verursacht nur minimale Kosten.

Doch der geringe Aufwand auf der Kostenseite kann das Verhalten der Software-Benutzer-Programmierer noch nicht erklären. Für rationale Akteure wäre es immer noch sinnvoller, als Trittbrettfahrer am Open-Source-Phänomen teilzunehmen, d.h. ohne eigene Leistungen von den Arbeiten anderer zu profitieren, statt aktiv an der Erzeugung solcher Software mitzuwirken. Für einen Beitragsleister muss in irgendeiner Form ein selektiver Vorteil existieren, ein Nutzen, den nur jene Personen geniessen können, die sich engagieren. Allerdings gilt in Situationen, in denen die Kosten einer Freigabe gering sind (*low-cost*-Situationen), dass solche selektiven Vorteile nicht mehr gross zu sein brauchen, damit für die Benutzer-Programmierer ein Engagement für Open-Source-Software vorteilhaft erscheint.

Wenn es nun gelingt, solche selektiven Vorteile, von welchen nur die Prosumer, nicht aber die Trittbrettfahrer profitieren können, zu identifizieren und zu quantifizieren, so ist das eingangs beschriebene Paradox im Wesentlichen aufgelöst.

3.5 Zur Motivation von Open-Source-Programmierern

Welche selektiven Vorteile könnten also für Benutzer-Programmierer relevant sein und diese zu einem Engagement für Open-Source-Software motivieren?[7]

In der wissenschaftlichen Literatur über Open Source finden sich etliche Analysen, die sich mit diesem Thema auseinandersetzen. Auf Grund dieser Untersuchungen konnte ein ganzes Bündel von Anreizen identifiziert werden. Die mittlerweile vorliegenden empirischen Studien über das Open-Source-Phänomen,[8] konnten die Existenz dieser unterschiedlichen Motivationen tatsächlich bestätigen. In den folgenden Kapiteln werden diese Motivationen genauer erläutert.

3.5.1 Gebrauch

Die einfachste Begründung dafür, dass sich ein Software-Entwickler für ein Open-Source-Projekt engagiert, ist, dass er die von diesem Projekt erzeugte Software gebrauchen kann. Dieses Motiv entspringt dem Prosumer-Modell: Der Akteur hat ein Problem, welches mit der geeigneten Software gelöst werden kann, und erzeugt die entsprechende Software oder passt eine existierende seinen Bedürfnisse

[7] Auf die verschiedenen Geschäftsmodelle, die mit Open-Source-Software möglich sind und ein Engagement in solche Software motivieren können, gehe ich an dieser Stelle nicht ein. Open-Source-Geschäftsmodelle sind relevant für Software-Firmen und nicht für die Prosumer.

[8] Z.B. Robles, Scheider, Tretkowski und Weber (2001), Jorgensen (2001), Hars und Ou (2001), Dempsey, Weiss, Jones und Greenberg (2002), Ghosh u. a. (2002), Krishnamurthy (2002), Bonaccorsi und Rossi (2003), Hertel u. a. (2003), Healy und Schussman (2003), Lakhani und Wolf (2003), Hemetsberger (2003).

an. In der Studie von Lakhani und Wolf (2003) erhielt dieses Motiv die grösste Zustimmung.

3.5.2 Reputation und Signalproduktion

Raymond hat in seinem letzten Essay „Homesteading the Noosphere“ (Raymond, 2000a) aus seiner viel beachteten „The Cathedral and the Bazaar“-Trilogie beschrieben, wie die Normen und Tabus der Open-Source-Bewegung in Bezug auf den Erwerb von Reputation wirken. Wie Raymond aufzeigt, sind *code forking*, d.h. das Aufsplitten der Code-Basis in inkompatible Versionen, und vor allem das Löschen der Namen der Beitragsleister aus den Kreditfiles der Applikationen streng verpönt. Die Sensibilität der Open-Source-Community gegenüber diesen Normen macht Sinn im Hinblick auf den Aufbau von Reputation, denn ohne diese Normen wäre es viel schwieriger nachzuvollziehen, welcher Beitrag von welcher Person kommt, und damit wäre der Aufbau von Reputation stark erschwert.

Lerner und Tirole (2001) haben diesen Zusammenhang unter dem Aspekt der Signalproduktion analysiert. Gemäss dieser Argumentation ermöglicht es die Kombination von Offenlegung des Quellcodes zusammen mit den spezifischen Normen der Open-Source-Community, dass die Beiträge der einzelnen Entwickler sehr gut verfolgt werden können. Dies hat zur Folge, dass der Status eines Entwicklers in einem Projekt ein genaues Abbild seiner Reputation ist und diese wiederum sehr transparent die Qualität und Quantität seiner Beiträge reflektiert. Auf diesem Hintergrund besteht nun die Möglichkeit, diese Reputation zu monetarisieren, z.B. durch ein interessantes Arbeitsangebot oder einen verbesserten Zugang zu Risikokapital. Der Arbeits- bzw. Risikokapitalmarkt ist primär an Talent und Leistung und nicht an Reputation interessiert. Das Talent einer Person ist aber für Personen, die nicht mit dem Fachgebiet vertraut sind, schwer einzuschätzen. Wenn allerdings die Reputation einer Person ein gültiger Indikator für deren Talent ist, kann Reputation im oben beschriebenen Sinn als Signal wirken: Aus dem Wissen über ein Open-Source-Projekt und der Position einer Person in diesem Projekt kann z.B. ein potentieller Arbeitgeber gültige Rückschlüsse auf das Talent der Person ziehen. Signalproduktion wirkt am stärksten, wenn die technische Herausforderung gross ist, wenn die relevante Öffentlichkeit (d.h. die Peergroup) technisch erfahren ist, zwischen guten und herausragenden Leistungen unterscheiden sowie Leistung und Können wertschätzen kann (Weber (2000), Franck und Jungwirth (2002a)). Diese Bedingungen sind im Falle von Open Source in hohem Mass gegeben.

3.5.3 Identifikation mit der Gruppe

Ist eine Person gut in eine Gruppe eingebunden und kann sie sich stark mit den Gruppenzielen identifizieren, so kann häufig ein grosses Engagement dieser Person für die Gruppenziele beobachtet werden (Kollock und Smith, 1996). Hertel u. a. (2003) untersuchten in einer empirischen Studie unter Linux-Entwicklern, ob dieses Phänomen auch im Open-Source-Bereich von Bedeutung ist. Sie haben die Entwickler über verschiedene Aspekte ihrer Tätigkeit befragt und diese Angaben mit den Werten über das Engagement korreliert. Mit statistischen Methoden gelang es ihnen tatsächlich nachzuweisen, dass ein signifikanter Anteil des Engagements

der Entwickler durch ihre Identifikation mit ihrer Entwickler-Gemeinschaft erklärt werden kann.

Dieses Resultat ist von Lakhani und Wolf (2003) bestätigt worden. In ihrem Hacker-Survey untersuchten sie, wie sich die Identifikation mit der Projektgruppe auf das zeitliche Engagement auswirkt, und konnten dabei einen signifikant positiven Effekt feststellen.

3.5.4 Lernen

Der Einsatz für ein Open-Source-Projekt kann auch unter dem Aspekt des Lernens erklärt werden. Der Einsatz von neuster Technologie zur Bewältigung eines komplexen Problems ist für alle Beteiligten mit einem Gewinn an Erfahrung verbunden. Der Wunsch, seine Fähigkeiten als Software-Entwickler zu verbessern, taucht sowohl in der FOSS-Studie von Ghosh u. a. (2002) wie auch im Hacker-Survey von Lakhani und Wolf (2003) und in der Untersuchung von Hars und Ou (2001) mit hohen Zustimmungsraten auf.

3.5.5 Altruismus

Ein Engagement für Open Source kann auch begründet werden mit einem Gefühl für das Wahre und Richtige, z.B. dass die Freiheit der Menschen mit quelloffener Software zusammenhängt und dieses Gut durch kommerzielle und proprietäre Software-Anbieter gefährdet ist. Dieser Standpunkt wird pointiert von Stallman und seiner Free Software Foundation (FSF) vertreten.

Um eine ähnliche Motivation handelt es sich, wenn ein Programmierer an einem Open-Source-Projekt beteiligt ist, weil er selbst viele Open-Source-Produkte verwendet und nun die Verpflichtung spürt, selbst etwas für die Open-Source-Bewegung zu tun.

Während das erste Motiv auf ein abstraktes Gefühl und eine ideologische Haltung verweist, referiert das zweite Motiv auf ein Gefühl des gerechten Gebens und Nehmens, einer allgemeinen Reziprozität im Verhältnis des Individuums zu einer Gruppe. Ökonomisch können die auf diese Weise motivierten Beiträge als Spenden interpretiert werden. Ob der Beitrag geleistet wird oder entfällt, hat für den Programmierer keine nachvollziehbaren Auswirkungen. Das Open-Source-Projekt, in welchem sich der Spender engagiert, profitiert aber von den Beiträgen dieser Person.

Gemäss der Studie von Lakhani und Wolf (2003) bezeichneten ungefähr ein Drittel der befragten Open-Source-Programmierer diese Motive als relevant für ihr Engagement. Interessant ist, dass Lakhani und Wolf mit einer Cluster-Analyse zeigen konnten, dass das ideologisch fundierte Motiv, Software müsse offen sein, tatsächlich sehr nahe beim Motiv liegt, welches das Engagement aus dem Gefühl der Reziprozität heraus begründet.

3.6 Spass und Open Source

Wie ich in Kapitel 4.2 ausführe, erachte ich das Entwickeln von Software als genuin autotelische Tätigkeit. Programmieren wird von den Software-Entwicklern

als tief befriedigend empfunden, weil sie vollständig in dieser Tätigkeit aufgehen können. Programmieren kann also für Software-Entwickler einen grossen unmittelbaren Nutzen haben, der in Anlehnung an Huizinga (2001) als *homo ludens payoff* bezeichnet werden kann.

Bitzer, Schrettl und Schröder (2004) haben ein interessantes spieltheoretisches Modell analysiert, mit welchem die private Bereitstellung eines öffentlichen Guts im Falle von Open-Source-Software erklärt werden kann. In diesem Modell vergleichen sie die einmaligen Kosten, welche die Produktion der Software für den Entwickler bedeuten, mit dem Nutzen, der sich aus der unmittelbar konsumierten Freude an der Tätigkeit (*homo ludens payoff*) und dem Verhältnis aus Geschenknutzen[9] der Software und Diskontrate ergibt. Es überrascht nicht, dass die Software dann, wenn die beiden Nutzenkomponenten die Kosten übersteigen, in jedem Fall produziert wird. Die Autoren können aber aufzeigen, dass selbst dann ein Nash-Gleichgewicht existiert, wenn die Kosten über dieser Schwelle[10] liegen, weshalb auch in solchen Fällen die Open-Source-Software erzeugt wird.

Aus der Bedingung für ein solches Nash-Gleichgewicht können die Autoren ein Profil des Open-Source-Entwicklers ableiten:

Die Open-Source-Software wird von jener Person erzeugt, $i)$ für welche die Applikation den höheren Gebrauchswert darstellt, $ii)$ welche den höheren Geschenknutzen aus der Code-Spende bezieht, $iii)$ welche einen längeren Zeitrahmen hat (d.h. jünger ist), $iv)$ welche eine niedrigere Diskontrate hat (d.h. geduldiger ist), $v)$ welche kleinere Kosten hat, um die Software zu entwickeln oder alternativ $vi)$ mehr Spass an dieser Tätigkeit hat (Bitzer u. a., 2004, S. 18).

In der spieltheoretischen Betrachtung von Bitzer et al. sind es die beiden Komponenten Spass und Geschenknutzen, welche das Phänomen erklären, dass Software in reichlichem Mass unter eine Open-Source-Lizenz angeboten wird. Spass am Programmieren reicht unmittelbar noch nicht aus, um diesen Sachverhalt zu erklären. Wie Bitzer et al. richtig bemerken, sind auch Spasssucher primär rational denkende und handelnde Personen. Wenn Programmieren Spass macht, dann ist es immer noch besser, mit dieser Tätigkeit Geld zu verdienen, als das Resultat der Arbeit zu verschenken. Erst der mögliche Geschenknutzen durch die Freigabe unter einer Open-Source-Lizenz kann das Phänomen vollständig erklären.

Ich bin allerdings der Ansicht, dass unter gewissen Bedingungen der Faktor Spass ausreichend ist, um das Engagement von Open-Source-Entwicklern zu verstehen. Spass als alleiniges Motiv für ein solches Engagement reicht aus, wenn erstens gezeigt werden kann, dass sich Programmieren in einem Open-Source-Projekt unterscheidet von der Tätigkeit eines Software-Entwicklers in einem kommerziellen Unternehmen, und zweitens, dass diese Unterschiede dazu führen, dass in einem Open-Source-Projekt mehr Spass erlebt werden kann als unter kommerziellen Bedingungen.

Es ist tatsächlich möglich, solche Unterschiede zu identifizieren. Das Zusammenspiel der in Tabelle 3.2 aufgeführten Punkte führt dazu, dass ein Open-Source-Projekt den Programmierern (abgesehen von der monetären Entlohnung) deutlich

[9] Unter dem Geschenknutzen subsumieren die Autoren die verzögerten Nutzenkomponenten, die für den Autor der Software entstehen können, z.B. Reputation, sozialer Status, Signalproduktion.

[10] D.h. $c_i \geq p_i + g_i/r_i$ wo c_i: Erzeugungskosten, p_i: homo ludens payoff, g_i: Geschenknutzen der erstellten Software, r_i: Diskontrate.

1. Der Projektmanager in einem kommerziellen Umfeld ist üblicherweise nicht diejenige Person, welche die *Vision* über die Applikation entwickelt hat und pflegt.
2. Der Projekteigentümer in einem Open-Source-Projekt hat keine *formale Autorität*.
3. Die Programmierer in einem Open-Source-Projekt können üblicherweise nicht über direkte *monetäre Anreize* zu einem Engagement oder zur Steigerung ihres Engagements bewegt werden.
4. Das Engagement in einem Open-Source-Projekt erfolgt üblicherweise über eine Selbstselektion. Entsprechend ist zu erwarten, dass die *Herausforderungen*, welche das Projekt den Programmierern bieten, aus der Sicht der Projektmitarbeiter *optimal* sind.
5. Ein Open-Source-Projekt hat üblicherweise keine *Abgabetermine*.

Tabelle 3.2: Unterschiede von kommerziellen zu Open-Source-Projekten

attraktivere Arbeitsbedingungen offerieren kann als der kommerzielle Bereich.

Weil der Projekteigentümer eine *Vision* des Projekts hat, ist es ihm viel eher möglich, die am Projekt Interessierten für das Projekt und seine Ziele zu begeistern. Da der Projekteigentümer *keine formale Autorität* hat, muss er mit sachlicher und sozialer Kompetenz die am Projekt Mitarbeitenden überzeugen. Weil die Programmierer ohne Kostenfolgen aus dem Projekt ausscheren können, muss der Projekteigentümer auf die Bedürfnisse der Beitragsleistenden eingehen, will er ihr Engagement nachhaltig nutzen.

Das Fehlen von *Abgabeterminen* ist ein wesentliches Merkmal, welches die Arbeitsbedingungen an einem Open-Source-Projekt kennzeichnet. Ein fixer Abgabetermin führt dazu, dass die Programmierer unter Zeitdruck arbeiten und eine Arbeit abliefern, welche oft nicht ihren Qualitätsanforderungen entspricht und sie aus diesem Grund unbefriedigt lässt (vgl. dazu Raymond (2000b), DeMarco und Lister (1987) oder Amabile, DeJong und Lepper (1976)). Weiter gilt, dass das Fehlen von Deadlines dem Projekteigentümer die Möglichkeit gibt, besser auf die Anregungen aus der Entwicklergemeinschaft und dem Benutzerkreis einzugehen.[11]

DeMarco und Lister (1987) haben diesen Aspekt vertieft erforscht. Sie analysierten zu diesem Zweck die Produktivität in verschiedenen kommerziellen Softwareprojekten und untersuchten dabei, ob und in welcher Form ein Zusammenhang zur Art von Zeitvorgaben erkannt werden konnte. Sie stellten fest, dass in Projekten ohne jeglichen Termindruck die Produktivität um rund 30% höher lag als in Projekten, in welchen der Zeitplan von einem neutralen und erfahrenen System-

[11] Scacchi (2001) zeigt in seiner Studie auf, dass Open-Source-Projekte ohne formalen Anforderungsspezifikations-Prozess auskommen, ohne dass solche Applikationen dadurch an Qualität einbüssen.

analysten festgelegt worden war. Verglichen mit Projekten, in welchen der Zeitplan von den beteiligten Entwicklern bzw. Projektmanagern erstellt wurde, lag der Produktivitätsvorteil sogar bei 50% bzw. 90%.[12] DeMarco und Lister erkennen einen engen Zusammenhang zwischen der Produktivität von Software-Entwicklern und dem Spass, den diese an ihrer Arbeit haben, wobei sie unter Spass das von Csikszentmihalyi beschriebene Flow-Gefühl verstehen. Für DeMarco und Lister wird hochproduktives und -kreatives Arbeiten ermöglicht durch das beinahe meditative Versunkensein, die euphorische Stimmung und den Verlust des Zeitgefühls, welche einen Flow-Zustand kennzeichnen. Auf diese Weise besteht eine logische Beziehung zwischen dem Fehlen von Abgabeterminen und einem Umfeld, welches lustvolles und damit auch produktives Arbeiten zulässt.

In Abschnitt 3.2 habe ich dargestellt, dass sich der Handlungsspielraum der verschiedenen Open-Source-Akteure in gewissen Bereichen unterscheidet. Warum ist dieser Sachverhalt relevant für meine Untersuchung? Wie in 3.2 definiert, hat der Projekteigentümer das Recht, die Lizenz der Software zu bestimmen. Im Extremfall hat der Projekteigentümer die Software schon erstellt, hat einen allfälligen *homo ludens payoff* schon konsumiert, bevor er sich entscheidet, diese Software unter einer Open-Source-Lizenz freizugeben. Eine solche Entscheidung kann mit Flow aber nicht erklärt werden. Die Lizenzierung von Software, ob nun als Open Source oder proprietär, erzeugt keinen Flow.

Anders sieht die Situation für einen Programmierer aus. Im Extremfall hat diese Person Freizeit und unterschiedliche Optionen, diese Freizeit zu verwenden. Für einen Programmierer ist es also relevant, welchen *homo ludens payoff* die jeweilige Handlungsoption verspricht.

Diese Ausführungen zeigen, dass Spass als singuläres Motiv in der oben dargestellten Weise nur das Engagement von Programmierer erklären kann. Für die Eigentümer von Open-Source-Projekten braucht es notwendigerweise andere oder zusätzliche Motive, damit deren Handlung verständlich wird.

Die Bedeutung von Spass als Motivation für ein Engagement im Open-Source-Bereich wird von den bisherigen empirischen Studien weitgehend bestätigt. Beispielsweise ergaben die Fragen nach Kreativität und Flow in der Studie von Lakhani und Wolf (2003) ausserordentlich hohe Zustimmungsraten von 61% respektive 73%.

3.7 Der Stand der Forschung

Einen wichtigen Impuls für die Erforschung des Open-Source-Phänomens gab Eric Raymond mit seiner Trilogie „The Cathedral and the Bazaar“[13]. Der aktuelle Stand dieser Forschung wird im Übersichtsartikel von Rossi (2004) recht akkurat wiedergegeben. In diesem Überblick unterteilt Rossi die Forschungsanstrengungen in die folgenden Richtungen: Untersuchungen über die Motivation der

[12] Das Sample der Untersuchung von DeMarco und Lister (1987, S. 28) besteht allerdings nur aus 103 Projekten, die Untersuchung ist nicht in statistischem Sinn repräsentativ.

[13] Die Trilogie wurde unter diesem Titel von O'Reilly publiziert (Raymond, 1999a). Die Trilogie besteht aus den Teilen „The Cathedral and the Bazaar“, „Homesteading the Noosphere“ und „The Magic Cauldron“ und kann von der Website des Autors (http://www.catb.org/∼esr/writings/) heruntergeladen werden.

Open-Source-Entwickler, Arbeiten über die Führung von Open-Source-Projekten und die Koordination der Beitragsleister, Betrachtungen über die Beziehung vom Open-Source- zum kommerziellen Software-Bereich, Analysen des Open-Source-Entwicklungsmodells in Bezug auf die intellektuellen Eigentumsrechte und Arbeiten, welche die Bedeutung der Unterstützung der öffentlichen Hand für Open-Source-Software untersuchen.

Die Diskussion um die Bedeutung von extrinsischen Motivationen zur Erklärung des Verhaltens der Beitragsleister ist stark vom Artikel von Lerner und Tirole (2001) geprägt worden. Die Autoren deuten in ihrer Arbeit das Engagement der Open-Source-Programmierer als Investition in die Reputation als herausragende Software-Entwickler. Diese Reputation kann in der Folge als Signal für den Arbeits- und Risiko-Kapitalmarkt eingesetzt werden (vgl. Abschnitt 3.5.2). Weber (2000) relativiert allerdings die Bedeutung dieses Motivationsfaktors: Wenn Reputation von grosser Wichtigkeit für die Motivation der Open-Source-Entwickler wäre, wo müsste ein harter Wettbewerb um Reputation zu beobachten sein, beispielsweise in Form von offenen Herausforderungen der Position des Projekteigentümers oder in Form von strategischem Forking. Solche Phänomene treten aber relativ selten auf, was darauf hinweist, dass die Bedeutung von Reputation und Signalproduktion beschränkt ist. Als weitere relevante extrinsische Motivationen sind die Benützer-Bedürfnisse (Franke und von Hippel, 2002), Leistungsverbesserungen sowie Lernen identifiziert worden (siehe Abschnitt 3.5.4).

Die Rolle der intrinsischen Motivation von Open-Source-Entwicklern ist von Praktikern (z.B. Raymond (1999b), Torvalds und Diamond (2001)) wie auch von Wissenschaftlern (z.B. Bitzer u. a. (2004), siehe Abschnitt 3.6) betont worden. Als Motivationen dieser Art kommen neben Spass am Programmieren auch die Zugehörigkeit zu einer Entwicklergemeinschaft sowie die Verpflichtung, gewissen Normen wie Altruismus oder generalisierter Reziprozität zu genügen, in Frage (Lakhani und von Hippel (2000), Kollock (1999), Zeitlyn (2003)). Etliche Untersuchungen (z.B. Franck und Jungwirth (2002b), Osterloh, Rota und Kuster (2002b)) weisen darauf hin, dass sich im Bereich „Open Source“ intrinsische und extrinsische Motivationen nicht ausschliessen, sondern im Gegenteil befruchten können. Während Franck und Jungwirth die Koexistenz unterschiedlich motivierter Beitragsleister auf Grund des institutionellen Arrangements, welches durch die Lizenz erzeugt wird, sichergestellt sehen (siehe unten), kommt gemäss Osterloh u. a. (2002b) die Rolle des Bindeglieds den intrinsisch motivierten, die sozialen Normen der Gemeinschaft beachtenden Entwickler zu. Diese Gruppenmitglieder sind wertvoll, weil sie einerseits am ehesten bereit sind, allgemeine Unterstützung anzubieten und dadurch ein Klima des Vertrauens zu schaffen, und andererseits die Bereitschaft haben, solche Personen zur Rechenschaft zu ziehen, welche die Gruppennormen verletzen. Solche Interventionen stellen ein öffentliches Gut innerhalb der Gruppe dar, sind aber wichtig, um das soziale Dilemma zweiter Ordnung zu entschärfen und damit das längerfristige Überleben der Gruppe sicherzustellen.

Lakhani und von Hippel zeigen am Beispiel der Apache-Usergruppe, wie die Unterstützung der Benützer der Apache-Websever-Software organisiert ist und welche Motivationen die Teilnehmer zu ihren Hilfeleistungen antreiben. Kritisch für die Bereitschaft für ein relativ profanes und wenig angesehenes Engagement, die Software-Benützer bei ihren Problemen zu unterstützen, sind die

verhältnismässig geringen Kosten einer solchen Hilfeleistung: Eine Frage wird beantwortet, wenn die Person die Antwort kennt, und die Antwort wird online publiziert. Als wichtigste Motive für ihre Hilfeleistung gaben die Informationsanbieter an, aus einem Gefühl der Reziprozität zu handeln, Open Source fördern zu wollen, Spass an der Beantwortung der Frage zu haben und ihre Reputation in der Open-Source-Community steigern zu wollen (Lakhani und von Hippel, 2000, S. 30). In seinem Essay vergleicht Kollock (1999) digitale Güter mit Geschenken und öffentlichen Gütern. Der Erzeugung von digitalen Gütern kommt zugute, dass deren Produktionskosten drastisch gesenkt werden, dass jede Information, welche online veröffentlicht wird, sofort zu einem öffentlichen Gut wird, und dass eine Person ausreicht, um ein solches öffentliches Gut zu erzeugen. Unter diesen Umständen können eher egoistische Motive wie die Antizipation von Reziprozität, der Wunsch nach Reputation oder nach Wirksamkeit ausreichen, um das Gut anzubieten. Weiter kann der Bedarf eines Gruppenmitglieds nach diesem Gut einen eher altruistisch motivierten Beitragsleister veranlassen, das Gut zu erzeugen. Auf Grund der geringen Kosten der Veröffentlichung ist es aber auch möglich, dass das Gut als Nebeneffekt veröffentlicht wird: Es ist aus anderen Gründen schon erzeugt worden und die Kosten der Freigabe sind vernachlässigbar. Zeitlyn verwendet kultur-anthropologische Begriffe und Konzepte, um Tausch-Phänomene im Open-Source-Bereich zu erklären. Er vergleicht Open-Source-Projekte mit Verwandtschaftsgruppen. In Verwandtschaftsgruppen werden Geschenke nicht unter dem Aspekt der Reziprozität gemacht, sondern mit der Absicht, symbolisches Kapital (z.B. Reputation) zu akkumulieren (Zeitlyn, 2003, S. 1289f).

Neben der Frage, warum und unter welchen Bedingungen sich Software-Entwickler an Open-Source-Projekten beteiligen, stellt auch die Frage, wie die unzähligen Beiträge der unterschiedlichen Entwickler erfolgreich koordiniert werden können, ein interessantes Untersuchungsgebiet dar. Zu diesem Bereich gehören Studien, welche die Merkmale des Open-Source-Entwicklungsmodells generell untersuchen (z.B. Raymond (2000b), Feller und Fitzgerald (2002), Scacchi, Jensen, Noll und Elliott (2005)). Feller und Fitzgerald entwickeln zu diesem Zweck ein Framework, um das Open-Source-Phänomen zu untersuchen. Gemäss diesem Rahmen wird Open Source analysiert, indem nach der Qualifikation (Was ist Open-Source-Software?) und der Transformation (Wie wird der Input, die Beitragsleistung, in stabile Software umgewandelt? Wie funktioniert die Koordination und Steuerung der Beitragsleister?) gefragt wird. Ebenso wichtig ist die Frage nach den Stakeholdern (Beitragsleistende, Benutzer, Firmen, Non-Profit-Organisationen), die alle in verschiedenen Rollen (d.h. als Klienten, Akteure und Eigentümer) auftreten können, und der technologischen Einbettung (Mailinglisten, Newsgruppen, webbasierte Diskussionsforen, Versionenverwaltung etc). Die letzte Komponente dieses Frameworks bildet die Weltanschauung, d.h. die technischen, ökonomischen und politischen Motivationen und Anreize auf Makro- und Mikro-Ebene (Feller und Fitzgerald, 2002, S. 58). Scacchi u. a. (2005) benutzten virtuelle ethnographische Techniken, um das wechselseitig verlinkte Netz von Online-Dokumenten und Projekt-Erzeugnissen, welche den Entwicklungsprozess eines Open-Source-Projekts charakterisieren und dokumentieren, zu analysieren und auf diese Weise Merkmale und Eigenheiten des Open-Source-Entwicklungsmodells zu beleuchten.

Andere Studien stellen die Entwicklergemeinde von Open-Source-Projekten

in den Mittelpunkt ihrer Untersuchung und arbeiten mit soziologischen oder ethnographischen Methoden. Ziel solcher Studien ist beispielsweise, eine Typologie von Open-Source-Programmierern zu entwickeln (z.B. Dempsey u. a. (2002), Kim (2003)), formelle und informelle Strukturen in Open-Source-Projekten zu identifizieren (Raymond (2000a), Crowston, Heckman, Annabi und Masango (2005), Wagstrom, Herbsleb und Carley (2005)) oder die Koordination der Beiträge der verschiedenen Beteiligten zu verstehen (z.B. Kuwabara (2000), Moon und Sproull (2000), Scacchi (2001), Dafermos (2001)). Dempsey u. a. (2002, S. 71) untersuchten beispielsweise die Linux-Entwicklergemeinde und stellten dabei fest, dass die Arbeit an Linux auf eine sehr grosse Anzahl Programmierer aufgeteilt ist. Kim (2003) analysierte in seiner Arbeit zwei erfolgreiche Open-Source-Projekte und identifizierte als Erfolgsfaktoren die Präsenz der Projekteigentümer und die Leichtgewichtigkeit des Entwicklungsprozesses, welcher situativ den Anforderungen angepasst werden kann. Raymond (2000a) beschreibt in seinem Essay „Homesteading the Noosphere“ die Gewohnheiten und Normen, welche Eigentum und Kontrolle in Open-Source-Projekten regeln. In seiner Darstellung ist es der Wettbewerb um Prestige und Reputation, welcher die Open-Source-Projekte antreibt. Die Normen in Open-Source-Projekten[14] stellen eine Art „Eigentumsrechte“ dar, welche in Analogie zur Landnahme im Lockschen Sinn verstanden werden können. Sie haben den Zweck, die Anerkennung der Peers derjenigen Person zukommen zu lassen, welcher diese Ehre gebührt (Raymond, 2000a, S. 15). Weil in diesem Reputationskampf die richtigen Personen belohnt werden, nämlich die produktiven, kreativen, innovativen, ist die Open-Source-Bewegung erfolgreich.

Crowston u. a. entwickelten ein konzeptuelles Modell für das Verständnis der Führung von Open-Source-Projekten. In diesem Modell ist eine erfolgreiche Projektführung das Resultat davon, dass es der Projekteigentümer versteht, die Komplexität des Programm zu reduzieren, indem er ein mentales Modell des Programms schafft, an welchem die Projektbeteiligten teilhaben können. Weiter ergibt sich die Projektführung, weil die unterschiedlichen Projektrollen unterschiedlichen Zugang zu den Projektressourcen bedeuten und die Normen und formalen Regeln und Prozesse im Projekt dem Eigentümer Legitimität geben (Crowston u. a., 2005, S. 14). Wagstrom u. a. setzten die Methoden der sozialen Netzwerkanalyse ein, um mit einer Untersuchung von Projekt-Mailinglisten die soziale Netzwerkstruktur der Open-Source-Communities zu beschreiben. Diese Untersuchung zeigte, dass die Verbindungen zwischen den Projekten enger waren als angenommen, während der Austausch innerhalb der Projekte unterhalb der Erwartungen lag. Mit dem Resultat aus der Netzwerkanalyse simulierten die Autoren die Entwicklung der Fitness von Open-Source-Projekten in Abhängigkeit der Fähigkeiten der Programmierer und der Stärke der gegenseitigen Abhängigkeit (Kopplung) der Programm-Merkmale (Features). Mit solchen Simulationen konnten sie zeigen, dass sowohl die Benützer wie auch die Entwickler rasch zu Projekten abwandern, welche fitter sind, d.h. von kompetenten Software-Entwickler schnell zu Produktionsreife gebracht werden. Je enger die unterschiedlichen Merkmale einer Applikation miteinander verhängt sind, desto stockender kommt das Projekt voran (Wagstrom u. a., 2005, S. 21).

[14] Raymond bezeichnet das Verbot des Löschens des Autorenvermerks bei Code-Modulen sowie die ablehnende Haltung gegenüber dem Abspalten von Projekten als solche Gewohnheiten.

Kuwabara (2000) erklärt den Erfolg des Linux-Projekts aus dem geglückten Zusammenspiel von Selbstorganisation der Entwicklergemeinde und der evolutionären Projektentwicklung. Der von Raymond beschriebenen Kampf um Reputation erzeugt dabei den Rückkopplungsmechanismus, welcher die Selbstorganisation des Open-Source-Projekts um den Projekteigentümer oder um andere Projektverantwortliche ermöglicht. Gleichzeitig führt der Kampf um Reputation dazu, dass die Grenzen (des technologisch Möglichen) erkundet werden, was die Evolution des Projekts antreibt. Moon und Sproull (2000) sehen den Erfolg des Linux-Projekts im Zusammenwirken der Modularisierung der Projektarbeit, der Anreizstruktur in Open-Source-Projekten sowie der sich durch die Arbeit und in der Arbeit ergebenden Praxisgemeinschaft (*community of practice*). Die Zerlegung des Systems in kleine, überschaubare Module reduziert den Koordinationsaufwand und macht damit auch grosse und komplexe Systeme (wie z.B. ein Betriebssystem) verwaltbar. Weiter gilt, vor allem in verteilten Projekten, wo die einzelnen Beitragsleister mit grosser Autonomie handeln, dass die Anreiz- und Motivationsstrukturen sorgfältig beachtet werden müssen. Damit sich in einem Open-Source-Projekt eine Praxisgemeinschaft entwickelt, braucht es einerseits elektronische Kommunikations-Mittel, andererseits differenzierte Rollen und Normen. Diese müssen in verteilten Teams explizit vermittelt werden.

Scacchi untersuchte in seiner Arbeit (Scacchi, 2001), wie sich Open-Source-Projekte bezüglich der Spezifikation der Anforderung an die zu entwickelnde Software von kommerziellen Projekten unterscheiden. Er identifizierte dabei acht Merkmale, mit welchen auf informelle Weise die Anforderungen in Open-Source-Projekten spezifiziert werden: 1) Die Anforderungen an die Software entstehen in einem Diskussionsprozess innerhalb der Entwicklergemeinde, 2) die Entwickler erzeugen und gebrauchen gmeinschaftliche mentale Konstruktionen der Erwartung an die Funktionsweise des Systems, 3) in Form von FAQs (*frequently asked questions*) und informellen Anleitungen werden in semi-strukturierter Form implizit die Benützer-Anforderungen festgehalten, 4) mit externen Publikationen (als Bücher oder in Zeitschriften) werden interessierten Software-Entwicklern die funktionalen und nicht-funktionalen Anforderungen der Applikation erklärt, 5) der öffentlich einsehbare Code stellt eine Implementierung der Anforderungen an die Applikation dar, während die Installationsbeschreibungen explizite Anforderungen an das Zielsystem, auf welchem die Applikation installiert werden soll, aufführen, 6) mit der Software, mit welcher Fehler und Probleme rapportiert und verfolgt werden (*bug reports/issue tracker*), werden Anforderungen erfasst, reartikuliert und verfeinert, 7) traditionelle Formen der Anforderungs-Spezifikation in Form von System-Dokumentationen, sowie 8) die Software-Erweiterungsmechanismen, welche gleichzeitig zur Herausbildung der Anforderungen an offene Software beitragen wie auch diese ermöglichen. In der Studie von Dafermos (2001) wird, am Beispiel des Linux-Projekts, ein allgemeiner Rahmen entwickelt, mit welchem das Management von virtuellen dezentralisierten Teams verstanden werden kann. Dafermos folgert in seiner Arbeit, dass solche Teams am produktivsten arbeiten, wenn eine intensive Interaktion sowie eine ungehinderter Zugang zu allen Informationen etabliert werden kann.

Ein weiteres wichtiges Thema für die Projektteams im Open-Source-Bereich betrifft die Frage, wie die organisatorischen Strukturen, die Kooperation und Koor-

dination im Projektverlauf aufrechterhalten werden können (z.B. Kollock (1999), Osterloh u. a. (2002b), Robles, Gonzalez-Barahon und Michlmayr (2005)). Kollock untersuchte das Linux-Projekt auf den Hintergrund des Problems von Kooperation und des kollektiven Handelns. In seiner Untersuchung stellt er die Hypothese auf, dass die Möglichkeit von Online-Interaktionen die Kosten und den Nutzen von sozialen Aktionen dramatisch ändern. Er kommt dabei zur Schlussfolgerung, dass Online-Gemeinschaften folgende fünf Kriterien erfüllen müssen, um stabil zu bleiben: 1) Die Interaktion muss kontinuierlich und dauerhaft erfolgen, 2) die Namen und Identitäten der Gruppenmitglieder müssen stabil bleiben, 3) das in früheren Interaktionen erarbeitete Wissen muss archiviert werden, damit in späteren Phasen darauf zurückgegriffen werden kann, 4) die Beiträge müssen sichtbar sein und der Einsatz der Beitragsleister anerkannt werden, 5) die Gruppengrenzen müssen definiert sein und verteidigt werden. Osterloh u. a. argumentieren, dass die Teilnahme von intrinsisch motivierten Programmierern eine notwendige Voraussetzung dafür ist, dass Open-Source-Projekte überleben können. Für Open-Source-Projekte gilt das Problem des kollektiven Handelns, d.h. es muss Personen in der Gruppe geben, welche bereit sind, den Gruppennormen Geltung zu verschaffen und die anderen Gruppenmitgliedern zu unterstützen. Solche Handlungen können aber nur von intrinsisch motivierten Personen, welche den Gruppennormen grosse Bedeutung zugestehen, übernommen werden (Osterloh u. a., 2002b, S. 15f). Robles u. a. gingen in ihrer Studie über das Debian-Projekt[15] der Frage nach, wie es einem Open-Source-Projekt gelingt, die Beitragsleistenden an das Projekt zu binden und wie die Projektaufgaben bei einem wechselnden Personenbestand aufrechterhalten werden können. In ihrer quantitativen Untersuchung konnten sie nachweisen, dass die Fluktuationsrate im Debian-Projekt relativ klein ist, insbesondere verglichen mit der Fluktuationsrate von Software-Entwicklern in kommerziellen Software-Firmen. Auch konnten sie zeigen, dass die Wahrscheinlichkeit, dass eine Aufgabe eines Programmierers, welcher das Projekt verlässt, von einem neuen Programmierer übernommen wird, hoch ist. Aus diesen Beobachtungen folgern die Autoren, dass es dem Debian-Projekt erstaunlich gut gelingt, den Freiwilligeneinsatz über die Zeit stabil aufrechtzuerhalten (Robles u. a., 2005, S. 107).

Open Source wird vielfach als Gegensatz zur kommerziellen, proprietären Softwareproduktion betrachtet oder postuliert. Wissenschaftliche Untersuchungen in diesem Gebiet zeigen aber, dass die Beziehungen zwischen dem Open-Source-Bereich und dem kommerziellen Bereich vielfältiger und komplexer sind. Einerseits steht Open-Source-Software im Wettbewerb mit proprietärer Software und es stellt sich die Frage, auf welcher Ebene Open-Source-Software proprietärer Anbieter herausfordern kann (z.B. Johnson (2001), Challet und Du (2003)). Auf der anderen Seite haben kommerzielle Software-Anbieter verschiedene Strategien und Geschäftsmodelle entwickelt, um Open-Source-Software produktiv für ihre Unternehmensziele einzusetzen (z.B. Hecker (1999), Leiteritz (2004), Gabriel und Goldman (2002)). Johnson beschreibt das Entwickeln von Open-Source-Software mit Hilfe eines systemtheoretischen Modells, welches aus dem Nutzen von realisierten Programmkomponenten für die Benützer-Programmierer, aus den Kosten für die Realisierung, aus den Annahmen über das Verhalten der anderen Program-

[15] Debian ist eine Linux-Distribution, welche vollständig auf Freiwilligenarbeit beruht.

mierer und aus der daraus abgeleiteten Strategie basiert. Auf Grund von Gleichgewichtsüberlegungen kann Johnson zeigen, dass die Überlegenheit des Open-Source-Entwicklungsmodells in gewissen Situationen verglichen mit proprietärer Software dadurch zustande kommt, da im Falle von Open Source der Talent-Pool des ganzen Internets ausgeschöpft werden kann. Allerdings kann Trittbrettfahren dazu führen, dass gewisse Projekte auch bei grossen Entwicklergemeinden nicht durchgeführt werden. Umgekehrt trägt Trittbrettfahren positiv dazu bei, dass die Häufigkeit von redundanten Leistungen limitiert wird (Johnson, 2001, S. 26). Challet und Le Du verglichen das Open-Source-Entwicklungsmodell mit proprietärer Software bezüglich der Geschwindigkeit, mit welcher die Fehlerquote im Programm reduziert wird. Mit einem stochastischen Modell der Fehlerdynamik konnten sie zeigen, dass das Open-Source-Modell besser abschneidet, d.h. schneller fehlerfrei wird, weil die Benützer allfällige Fehler immer der neusten Version melden. Bei proprietärer Software dagegen melden die Benützer die Fehler der aktuell verfügbaren Version, während die Entwickler intern den Code schon weiterentwickelt hat auf einen Stand, welcher nicht mehr mit demjenigen des Benützers übereinstimmt (Challet und Du, 2003, S. 3).

Hecker hat als Mitarbeiter von Netscape die Freigabe des Mozilla-Quellcodes mitgestaltet und schildert in seinem Essay, wie es Netscape mit ihrer Open-Source-Strategie gelungen ist, die gestiegenen Herausforderungen an einen wachsenden Software-Anbieter zu meistern. Damit eine bestehende Applikation unter einer Open-Source-Lizenz freigegeben werden kann, muss neben der Wahl einer geeigneten Lizenz eine aus folgenden Punkten bestehende Strategie umgesetzt werden: 1) Der freizugebende Code muss durch geeignete Modularisierung von anderem Code sauber getrennt werden, 2) falls der Code Technologie von Dritt-Anbietern enthält, so muss solcher Code speziell behandelt werden, 3) der Code muss von internen Kommentaren etc. gesäubert werden, 4) der Produkt-Entwicklungsprozess muss so umgestaltet werden, dass der Einbezug einer Open-Source-Entwicklergemeinde möglich ist (Hecker, 1999, S. 50). Leiteritz beschreibt in seiner Untersuchung (Leiteritz, 2004) unterschiedliche Geschäftsmodelle mit Open-Source-Software. Der Autor identifizierte folgende auf Open Source basierende Geschäftsmodelle: 1) Distributor von Open-Source-Software, 2) Anbieter von Open-Source-Applikationen, 3) Anbieter von elektronischen Geräten, welche mit Open-Source-Software gesteuert werden, 4) Dienstleister mit Open-Source-Software, 5) Mediator von Open-Source-Software. Jedes dieser Geschäftsmodelle analysiert Leiteritz unter dem Aspekt der Marktpositionierung, des Gewinnmusters, der Positionierung in der IT-Wertschöpfungskette, der strategischen Absicherung sowie der notwendigen betrieblichen Organisation. Gemäss Gabriel und Goldman (2002) kann das Sponsoring eines Open-Source-Projekts durch einen kommerziellen Software-Anbieter ein grosses Potential darstellen, wenn dieser eine „Innovation Happens Elsewhere“-Strategie verfolgt. Diese Strategie besteht darin, die Entwickler- und Benützergemeinde um ein Open-Source-Projekt zu Innovationen zu animieren, welche für die Entwicklung eigener, kommerzieller Produkte ausgewertet werden können.

Open Source ist ein öffentliches Gut. Bei jedem öffentlichen Gut stellt sich das Problem sowohl von Übernutzung wie auch von Unterversorgung, wodurch die nachhaltige Bereitstellung des öffentlichen Guts gefährdet wird. Da Software zu

praktisch verschwindenden Kosten kopiert werden kann, besteht bei Software keine Gefahr der Übernutzung. Hingegen bleibt die Frage, warum Unterversorgung durch Open-Source-Software anscheinend kein Problem darstellt. Zu dieser Frage konnten einige Studien mit Hilfe von spieltheoretischen Ansätzen interessante Hinweise liefern. So war es Bitzer u. a. (2004) möglich, den Spass am Programmieren als *homo ludens payoff* zu modellieren und auf diese Weise die ausreichende Bereitstellung von Open-Source-Software zu erklären. Bessen zeigte (in Bessen, 2001), dass das Open-Source-Entwicklungsmodell unter der Bedingung asymmetrischer Informationen dem proprietären Entwicklungsmodell überlegen ist, wenn es darum geht, ein komplexes Gut zu erstellen. Komplexe Güter sind durch eine Vielzahl von Möglichkeiten gekennzeichnet, die alle miteinander kombiniert werden können. Ein Software-Benützer ist jeweils nur an einer spezifischen Kombination dieser Möglichkeiten interessiert. Trotzdem muss ein kommerzieller Anbieter von komplexer Software jede dieser Kombinationen auf deren Funktionsfähigkeit testen, da ihm die Information fehlt, welche Kombinationen tatsächlich nachgefragt werden. Weist eine Software-Applikation einen zu hohen Komplexitätsgrad auf, so kann sie auf Grund der Vielzahl der Möglichkeiten und ihrer Kombinationen nicht mehr getestet werden. Ein kommerzielle Anbieter stoppt in diesem Fall die Produktion solcher Software oder reduziert ihren Funktionsumfang. Die Benützer-Programmierer von Open-Source-Software sind dagegen nur an jeweils einer relevanten Kombination der Möglichkeiten der komplexen Software interessiert. Solche Prosumer entwickeln, testen und debuggen genau diejenige Kombination, die für sie von Bedeutung ist. Solche Programmierer sind deshalb in der Lage, beliebig komplexe Software zu entwickeln. Eckert, Koch und Mitlöhner (2005) untersuchten das Angebot von Open-Source-Software unter dem Aspekt der Kooperation der verschiedenen Beitragsleister. Sie modellierten die Evolution von Kooperation in Open-Source-Projekten als wiederholtes Gefangenen-Dilemma. Die Züge in diesem Spiel sind gegeben durch Kooperation in Form von erhöhtem Engagement und Defektion in Form eines verringerten Engagements. Die Auszahlung erfolgt in Form der funktionsfähigen Applikation, als Runde wird beispielsweise ein Release der Software oder ein Arbeitsintervall verstanden. Auf diesen Annahmen kann nun die Entwicklung einer Population von stochastischen reaktiven Strategien simuliert werden. Mit solchen Simulationen konnten die Autoren zeigen, dass ohne zusätzliche Annahmen nach rund 200 Runden die Kooperations-Strategie Tit-for-Tat das Bild bestimmt. Diese wird nach weiteren 200 Runden durch eine grosszügiges Tit-for-Tat (*generous Tit for Tat*) abgelöst.[16]

Weitere Untersuchungen zum Aspekt von Open Source als öffentliches Gut behandeln die verschiedenen Facetten von Open-Source-Lizenzen. Die einen befassen sich mit der Frage des Zusammenhangs von Open-Source-Lizenzen und dem Schutz des intellektuellen Eigentums (z.B. Wiebe (2004), González (2005)), andere untersuchen die Auswirkung von Open-Source-Lizenzen auf die Innovationsfähigkeit von Open-Source-Projekten und des Software-Bereichs generell (z.B.

[16] *Tit for Tat* ist eine Strategie, bei welcher Kooperation der anderen Partei in der Vorrunde immer mit Kooperation und Defektion in der Vorrunde immer mit Defektion beantwortet wird. *Generous Tit for Tat* heisst eine Strategie, welche Kooperation der anderen Partei ebenfalls belohnt, wohingegen Defektion in der Vorrunde mit einer gewissen Wahrscheinlichkeit (z.B. 30%) trotzdem noch mit Kooperation beantwortet wird.

Bessen und Maskin, 2000), während weitere die Folgen von Open-Source-Lizenzen auf die Organisation von Open-Source-Entwicklergemeinden erörtern (z.B. Franck und Jungwirth (2002b), Lerner und Tirole (2002), siehe Abschnitt 3.2). Wiebe schildert in seiner Untersuchung die Geschichte der Softwarepatentierung und die Implikationen, welche die Möglichkeit von Softwarepatenten auf Open-Source-Software haben könnte. Seiner Ansicht nach besteht die grösste Gefahr für Open-Source-Software, wenn die Möglichkeit zur Patentierung von Software breit gefasst wird und insbesondere die Patentierung von Algorithmen möglich macht. Aber auch die Rechtsunsicherheit, die bei der Einführung von Softwarepatenten entsteht, kann sich negativ auf die Open-Source-Bewegung auswirken (Wiebe, 2004, S. 292).

González beschreibt in seinem Papier die Herausforderungen, welchen die Open-Source-Bewegung auf Grund des Rechtsstreits *SCO vs. IBM*[17] sowie der Patentierung von Software gegenüberstehen. Der Autor kommt dabei zum Schluss, dass die beschriebenen Herausforderungen Unsicherheit auslösen, die Open-Source-Bewegung aber keine Möglichkeit hat, durch irgendwelche Aktionen eine Klärung herbeiführen zu können, sondern dass eine Klärung nur durch entsprechende Entscheidungen vor Gericht erreicht werden kann (González, 2005, S. 218). Bessen und Maskin gehen in ihrer Studie der Frage nach, warum es Firmen in Bereichen, die wie der Software-Markt durch schwachen Patentschutz gekennzeichnet sind, trotzdem gelingt, innovativ zu sein. Die Autoren argumentieren, dass der Wettbewerb die zukünftigen Unternehmensgewinne steigern kann, wenn die Innovationen sequentiell und komplementär sind.[18] Die Firmen können damit die kurzfristig entgangenen Gewinne kompensieren. Ein Patentschutz in solchen dynamischen Bereichen führt dagegen zu einem Rückgang der Gesamtheit der Innovationen und dadurch zu einer Verringerung des sozialen Nutzens (Bessen und Maskin, 2000, S. 2).

Open-Source-Projekte sind dadurch, dass sie den Quellcode offenlegen, gewissen Risiken ausgesetzt: Die Software kann kommerziell angeeignet werden, das Projekt kann in inkompatible Versionen zerfallen (*forking*), das Projekt kann daran scheitern, eine Entwicklergemeinde aufzubauen etc. Mit der Wahl der geeigneten Lizenz kann in einem gewissen Rahmen auf diese Gefahren reagiert werden. Franck und Jungwirth (2002b) zeigen, wie die Wahl einer restriktiven Lizenz (z.B. GPL) eine Governance-Struktur bewirkt, welche sowohl die Interessen von Spendern wie auch diejenigen von Rentesuchern (d.h. Programmierern, die einen Nutzen aus dem Projektengagement ziehen, z.B. unmittelbar als Spass oder langfristig als Reputation) unter einen Hut bringt. Die Copyleft-Klausel einer restriktiven Lizenz verhindert, dass eine Code-Spende in einem abgeleiteten Werk proprietär angeeignet werden kann. Dieser Umstand macht Copyleft nicht nur für Spender, sondern auch für solche Entwickler interessant, die in ihre Reputation investieren, da jeder Codeteil, der als Folge proprietärer Aneignung aus dem Blickfeld der

[17] Im Jahr 2003 klagte die SCO Group IBM wegen Verletzung des intellektuellen Eigentums an: IBM habe der SCO gehörenden Unix-Code illegitimerweise in den Linux-Kernel überführt.

[18] Innovationen sind „sequentiell“, wenn die eine Erfindung auf einer vorangegangenen aufbaut. Unter „komplementären“ Innovationen verstehen die Autoren den Umstand, dass jeder potentielle Innovator einen anderen Forschungsansatz wählt, was die Gesamtwahrscheinlichkeit erhöht, dass die Erfindung innerhalb einer bestimmten Zeit tatsächlich realisiert wird.

Öffentlichkeit verschwindet, die Basis schmälert, auf welcher die Reputation solcher Software-Entwickler beruht. In einer quantitativen Analyse der Lizenzen von Open-Source-Projekten, die auf SourceForge registriert sind, haben Lerner und Tirole (2002) untersucht, von welchen Kriterien sich die Projekte bei der Wahl ihrer Open-Source-Lizenz leiten lassen. In ihren statistischen Auswertungen konnten sie zeigen, dass restriktive Lizenzen in Projekten untervertreten sind, welche in einer kommerziellen Umgebung stattfinden oder deren Applikationen auf proprietären Plattformen laufen. Restriktive Lizenzen sind stark in solchen Projekten vertreten, die sich an Endbenützer und Systemadministratoren ausrichten, wohingegen Projekte, die sich an Entwickler wenden, von restriktiven Lizenzen weniger Gebrauch machen. Ebenfalls stark vertreten sind restriktive Lizenzen in Applikationen, welche konsumorientiert sind (z.B. Spiele), während Applikationen, die auf den Software-Entwicklungsprozess ausgerichtet sind, eher liberale Lizenzen haben.

Ein letzter Bereich der wissenschaftlichen Forschung zum Thema „Open Source“ beschäftigt sich mit dem Verhältnis der öffentlichen Hand zu diesem Software-Entwicklungsmodell. Die wissenschaftlichen Untersuchungen in diesem Bereich befassen sich mit der Frage, wie gross die Verbreitung von Open-Source-Software in der staatlichen Verwaltung ist, ob die öffentliche Hand die Produktion von Software unter einer Open-Source-Lizenz unterstützen soll und wenn ja, in welcher Form (z.B. Hahn (2002), Schmidt und Schnitzer (2003), Comino und Manetti (2003), Gutsche (2005)). In seinem Sammelband zur Frage einer möglichen Regierungspolitik zu Open Source versammelt Hahn die Ansicht der vier Experten J. Bessen, D.S. Evans, L. Lessig und B.L. Smith zu diesem Thema. Die Schlüsselfrage, welche eine mögliche Intervention der öffentlichen Hand rechtfertigen kann, ist, ob im Software-Bereich ein Marktversagen vorliegt. Bessen sieht ein solches Marktversagen auf Grund von unvollständigen Verträgen und asymmetrischen Informationen. Dies rechtfertigt gemäss Bessens Ansicht aber keine Intervention der öffentlichen Hand, da dieses Problem durch Open Source gelöst werden kann. Allerdings sollten die Bestimmungen bezüglich Software-Patentierung zugunsten von Open-Source-Software restriktiver gefasst werden. Evans dagegen sieht keinerlei Marktversagen im Software-Bereich. Er empfiehlt als Strategie, dass sich die öffentliche Hand wie ein normaler Marktteilnehmer verhalten und Software für die Verwaltung auf Grund von Kosten-Nutzen-Betrachtungen auswählen soll. Wo sich die öffentliche Hand dafür entscheidet, Forschung und Entwicklung im Open-Source-Bereich zu subventionieren, so sollen die Ergebnisse solcher Investitionen unter liberalen Lizenzen freigegeben werden. Lessig empfiehlt als Regierungsstrategie ein klares Eintreten für offene Standards und Plattformen. Hat die öffentliche Hand die Wahl zwischen zwei Systemen, die bezüglich den übrigen Anforderungen gleichwertig sind, sich nur in der Lizenz unterscheiden, so soll sich die öffentliche Hand für Open Source entscheiden. Smith erkennt ebenfalls kein Marktversagen und ist deshalb der Ansicht, dass der Software-Markt die beste Kombination an Anreizen und Flexibilität bietet, um die Kundenwünsche zu befriedigen. Bei Interventionen der öffentlichen Hand besteht die Gefahr, dass dieses Zusammenspiel gestört wird. Smith sieht ein Rolle der öffentlichen Hand darin, solche Forschung im Software-Bereich zu fördern, deren Resultate unter einer liberalen Lizenz zur Verfügung stehen, damit diese in der Folge kommerziell aus-

gewertet werden können (Hahn, 2002, S. 7f).

Schmidt und Schnitzer (2003) analysieren in ihrem Beitrag die Bedingungen für Innovationen im Software-Bereich. Ihrer Meinung nach ist es das Profitmotiv, welches die Interessen von Software-Entwicklern und der Gesellschaft in Übereinstimmung bringt. Aus diesem Grund stehen sie den Anstrengungen der öffentlichen Hand im Open-Source-Bereich kritisch gegenüber. Direkte Subventionen der öffentlichen Hand für Open-Source-Projekte sowie eine Strategie, welche den Ersatz von proprietärer Software in der öffentlichen Verwaltung durch Open-Source-Software vorsieht, lehnen sie ab. Als vorteilhaft erachten die Autoren eine Politik, die sich nicht auf konkrete Open-Source-Projekte bezieht, sondern die Entstehung von offenen Standards und die Kompatibilität von Software-Applikationen födert.

Comino und Manetti setzen mathematischen Methoden ein, um den Wettbewerb zwischen Open-Source- und proprietärer Software zu analysieren. Das Ziel ihrer Untersuchung ist, die Wohlfahrts-Effekte unterschiedlicher Strategien der öffentlichen Hand abzuschätzen. Zu diesem Zweck verwenden sie ein statisches Modell mit zwei konkurrierenden Akteuren „Open Source“ und „proprietäre Software-Hersteller“. Als Resultat ihrer Berechnungen kommen die Autoren zum Schluss, dass eine Subventionierung von Open-Source-Benützern immer einen negativen Effekt hat, während sich Kampagnen zur Reduktion des Informations-Defizits bezüglich Open Source wohlfahrtssteigernd auswirken. Vorschriften, welche den Einsatz von Open-Source-Software in der Verwaltung erzwingen, sind nur dann wohlfahrtssteigernd, wenn die Anzahl der informierten Benützer genügend gross ist. Das Schlüsselelement im Modell von Comino und Manetti ist die Grösse der Bevölkerungsgruppe, die über Open Source informiert ist, da der Wettbewerb zwischen Open-Source- und proprietärer Software nur innerhalb der informierten Personen stattfindet (Comino und Manetti, 2003, S. 24). Gutsche (2005) analysiert den Wettbewerb zwischen Open-Source- und proprietärer Software ebenfalls mit Hilfe eines Duopol-Modells. Der Autor untersucht in seinem Papier, welche Auswirkungen eine nachfrage- oder angebotsorientierte Politik der öffentlichen Hand bezüglich Open Source unter den Bedingungen von Netzwerkeffekten, Kosten des Systemwechsels und des Systemwettbewerbs haben. Generell findet der Autor keinen Hinweis für einen wohlfahrtssteigernden Effekt einer Intervention der öffentlichen Hand. Hingegen kann sich eine Subventionierung von Open Source durch die öffentliche Hand wohlfahrtssteigernd auswirken, wenn Open-Source- und proprietäre Software im Wettbewerb zueinander stehen. Durch eine Unterstützung von Open Source kommt ein positiver Effekt im Sekundärmarkt von komplementären Gütern (Systemerweiterungen, Dienstleistungen etc.) zustande, welcher die Kosten einer solchen Intervention übertrifft.

Empirisch wurde das Phänomen „Open Source“ sowie die Motivation der Open-Source-Entwickler in den Studien von Robles u. a. (2001), Jorgensen (2001), Hars und Ou (2001), Dempsey u. a. (2002), Ghosh u. a. (2002), Krishnamurthy (2002), Bonaccorsi und Rossi (2003), Hertel u. a. (2003), Healy und Schussman (2003), Lakhani und Wolf (2003), Hemetsberger (2003) und Spaeth (2005) untersucht (siehe Tabelle 3.3).

In ihrer WIDI-Studie (*Who is doing it*) befragten Robles u. a. Open-Source-Entwickler weltweit nach dem persönlichen und beruflichen Hintergrund ihres Open-Source-Engagements. Sie fanden, dass 80% der Open-Source-Entwickler

einen professionellen Hintergrund im IT-Bereich haben und 33% im universitären Bereich tätig sind. Die Entwickler setzen im Durchschnitt 11.5 Stunden pro Woche für Open-Source-Software ein. 21% der Open-Source-Entwickler werden für ihre Tätigkeit bezahlt. 46% der Open-Source-Entwickler haben vom Engagement im Open-Source-Bereich beruflich profitiert, beispielsweise verdanken 15% ihren aktuellen Arbeitsplatz diesem Engagement und 2% ein Lohnerhöhung. Rund die Hälfte derjenigen, die bis anhin keinen beruflichen Nutzen aus ihrem Open-Source-Engagement ziehen konnten, erwarten einen zukünftigen Nutzen daraus. Der Frauenanteil in der WIDI-Studie betrug 1.4%.

Jørgensen untersuchte in seiner empirischen Arbeit (Jorgensen, 2001) die motivationalen und produktiven Auswirkungen der speziellen Release-Philosophie im FreeBSD Open-Source-Projekt. Bei FreeBSD handelt es sich um ein Unix-Derivat mit langer Open-Source-Tradition. Speziell an diesem Projekt ist, dass alle Änderungen am Code, ob es sich um Erweiterungen oder um Fehlerkorrekturen handelt, auf die gleiche Release-Version in der Versionenverwaltung appliziert werden.[19] Der Autor befragte 72 Entwickler des FreeBSD-Projekts mit Hilfe eines Online-Fragebogens sowie in Einzelinterviews. Als Resultat seiner Untersuchung konnte Jørgensen zeigen, dass sich die inkrementelle Vorgehensweise *see bug, fix bug, see bug fixed in new release* für die Entwickler stark motivierend auswirkt. Negativ schlägt allerdings zu Buche, dass die inkrementelle Vorgehensweise mit nur einer Code-Version das Entwickeln von neuer komplexer Programm-Funktionalität erschwert. Weiter zeigte die Untersuchung, dass 43% der Befragten für ihre Arbeit am FreeBSD-Projekt ganz oder teilweise bezahlt werden. 86% der Befragten geben an, dass ihre Beiträge einen Review erhalten haben. Gemäss 57% der Befragten haben die Reviews der eigenen Beiträge zu einer deutlichen Verbesserung der Fähigkeiten als Entwickler geführt.

Hars und Ou (2001) befragten 81 Programmierer[20] in Open-Source-Projekten zur Motivation ihres Engagements. 16% der Open-Source-Entwickler in dieser Studie werden für ihr Open-Source-Engagement bezahlt. Diese Entwickler tragen 38% der insgesamt für Open Source eingesetzten Zeit bei. Gemäss dieser Studie bewerten 80% der Befragten Selbstbestimmung hoch bis sehr hoch ein, für 88% ist die Bildung von Humankapital von grosser Bedeutung. Interessant ist, dass Selbstbestimmung deutlich wichtiger ist für Programmierer, die in ihrer Freizeit an Open-Source-Projekten arbeiten. Von diesen schätzen 93% die Selbstbestimmung als wichtig ein, während für bezahlte Open-Source-Programmierer mit 62% Zustimmungsrate dieser Faktor unterdurchschnittlich abschneidet. Für Studierende und Hobbyprogrammierer ist mit 97% Zustimmungsrate der Effekt der Bildung von Humankapital überdurchschnittlich wichtig.

Das Ziel der Studie von Dempsey u. a. (2002) war, ein Profil der Open-Source-Entwickler zu erstellen. Zu diesem Zweck analysierten sie die *Linux Software Maps (LSMs)* des Linux-Projekts.[21] Mit diesen Analysen konnten die Autoren zei-

[19] Bei Software-Projekten ist üblich, dass vor einem Release ein *feature freeze* stattfindet. Bei einem Code-Bestand unter *feature freeze* werden nur noch Fehlerkorrekturen angebracht. Damit aber die Entwicklung im Software-Projekt trotzdem weitergehen kann, werden in der Versionenverwaltung spezielle Entwicklungs-Versionen (*development branches*) geführt.

[20] Bei insgesamt 389 angeschriebenen Personen entspricht dies einer Rücklaufquote von 21%.

[21] Mit den *Linux Software Maps* geben die Beitragsleister Metadaten (z.B. Titel und Beschreibung

gen, dass die Beiträge zu Linux sehr breit gestreut waren: Von den insgesamt 2429 registrierten Entwicklern leisteten 91% 1-2 Beiträge, 2.2% mehr als 5 Beiträge. Nur von 13 Entwicklern stammten mehr als 10 Beiträge.

Mit einer breit angelegten Studie konnten Ghosh u. a. (2002) ein Profil der Open-Source-Entwickler entwerfen. Gemäss dieser Untersuchung sind die Open-Source-Entwickler im Durchschnitt 27 Jahre alt, wobei 25% der Programmierer älter als 30 Jahre sind. 71% der Open-Source-Entwickler kommen aus Europa (Anteil Frankreich: 16.5%, Deutschland: 12.4%), 13% aus Nordamerika (Anteil USA: 10.3%). 54% der Entwickler ziehen einen direkten oder indirekten Nutzen aus ihrem Open-Source-Engagement: 16% werden für ihre Open-Source-Entwicklungsarbeit bezahlt, 18% erhalten Geld für administrative Tätigkeiten für Open Source, 12% werden für andere Unterstützungsarbeiten für Open Source bezahlt. 17.5% der Open-Source-Entwickler konnten indirekt von ihrem Open-Source-Engagement profitieren, weil sie ihren Arbeitsplatz dank diesem Engagement fanden. Die Entwickler setzen durchschnittlich 12 Stunden pro Woche für Open Source ein. Mehr als 70% arbeiten weniger als 10 Stunden pro Woche für Open Source, 23% sogar weniger als 2 Stunden. 14% engagieren sich zwischen 10 und 20 Stunden, 9% zwischen 20 und 40 Stunden, immerhin noch 7% mehr als 40 Stunden. Als Motive für ihr Engagement geben 80% der Entwickler an, dass sie mit ihrem Einsatz neue Fähigkeiten erwerben wollen, 50% wollen ihr Wissen teilen, 33% sind durch die Kooperation in Open-Source-Projekten motiviert, ebenfalls 33% wollen mit ihrem Einsatz bestehende Open-Source-Software verbessern, und 30% wollen mit ihrem Engagement Open-Source-Software als öffentliches Gut unterstützen.

In seiner Untersuchung von SourceForge-Projekten ging Krishnamurthy (2002) der Frage nach, welche Merkmale aktive Open-Source-Projekte aufweisen. In dieser Analyse kam der Autor zum erstaunlichen Resultat, dass die 100 erfolgreichsten SourceForge-Projekte im Durchschnitt eine Gruppen-Grösse von 6.6 Entwicklern (Median: 1 Entwickler) haben. Den Erwartungen entsprechend gilt, dass ein Projekt umso mehr Entwickler hat, je mehr es beachtet wird (Korrelations-Koeff.: 0.56) und desto mehr Downloads es aufweist (Korrelations-Koeff.: 0.27).

Untersuchungsgegenstand der Studie von Bonaccorsi und Rossi (2003) war die Motivation von Software-Firmen, sich im Open-Source-Bereich zu engagieren. Zu diesem Zweck befragten sie 146 italienische Software-Firmen nach ihrer Motivation für ein solches Engagement. Die befragten Firmen bewerteten auf einer fünfpunktigen Likertskala (Wertung von 1-5) die ökonomischen Motive am höchsten (3.56), vor den technologischen (3.51) und den sozialen (3.39).

In ihrer Studie wollten Hertel u. a. (2003) untersuchen, wie sich die Gruppenidentität im Open-Source-Projekt auf das Projekt-Engagement auswirkt. An dieser Studie nahmen 69 Linux-Kernel-Entwickler und 72 Teilnehmer der Linux-Kernel-Mailingliste teil. In der Studie wurden verschiedene Motivationsfaktoren (Identifikation mit dem Projekt, Spass, Problemlösung etc.) mit einer fünfpunktigen Likertskala (Wertung von 1-5) erhoben und die Antworten mit dem Engagement für das Projekt korreliert. Die Probanden in dieser Untersuchung verwenden durchschnittlich 12.5 Stunden pro Woche für Open Source, wobei der Arbeitseinsatz von Kernel-Entwickler mit 18.4 Stunden deutlich höher ausfällt als derjenige der

des Beitrags, Datum etc.) zu ihren Codebeiträgen an.

Mailinglisten-Teilnehmer (6.6 Stunden pro Woche). Mit einem Wert von 4.6 erhielten die hedonistischen Teilnahmemotive die höchste Zustimmungsrate. Den grössten Erklärungsgehalt für das Engagement der Entwickler hatten pragmatische Motive, gefolgt von der Identität als Mitglied im Projekt und der Toleranz gegenüber dem Zeitaufwand, den das Open-Source-Engagement bedingt.

Healy und Schussman (2003) wiederholten die Untersuchung von Krishnamurthy (2002) mit einem weitaus grösseren Sample. Ihnen standen die Daten von 46'356 Projekten auf SourceForge zur Verfügung. Das Ergebnis deckte sich aber weitgehend mit der vorangegangenen Studie: Die Projektgrösse beträgt im Mittel 1.7 Entwickler (Median: 1 Entwickler, 95 Perz.: 5 Entwickler). Die Anzahl Downloads beträgt im Durchschnitt 2289 (Median: 0, 90 Perz.: 872). Diese Untersuchung bestätigt eindrücklich die schiefen Verteilungskurven (*power laws*), welche im Open-Source-Bereich anzutreffen sind.

Lakhani und Wolf befragten 648 Open-Source-Entwickler nach ihren Motiven zur Teilnahme in Open-Source-Projekten (Lakhani und Wolf, 2003). Die in dieser Studie befragten Entwickler verwenden im Durchschnitt 14 Stunden pro Woche für die Entwicklung von Open-Source-Software. Gefragt nach den kreativsten Momenten in ihrem Leben verglichen mit der Kreativität, welche sie im Open-Source-Projekt erleben, schätzten 61% der Befragten das im Open-Source-Projekt empfundene Mass an Kreativität als gleichwertig oder höher ein. 73% erleben häufig oder immer Flow-Zustände bei ihrer Open-Source-Tätigkeit. Als Motivation für das Engagement gaben die Entwickler zu 59% den Gebrauch an, zu 45% Spass, zu 41% die Möglichkeit des Lernens und zu 33% altruistische Motive. Um die Determinanten für den Open-Source-Einsatz zu bestimmen, wurden die verschiedenen Motive mit dem zeitlichen Engagement der Beitragsleister korreliert und der Effekt der einzelnen Beiträge mittels Regressions-Analyse berechnet. Dabei zeigte sich, dass die empfundene Kreativität (standardisierter Koeffizient: 1.6), die Bezahlung (0.9), die Freude an der Teamarbeit (0.8) sowie die Reputation (0.6) signifikant positiv beitragen, während die Arbeit in anderen Open-Source-Projekten (-1.6) und die IT-Ausbildung des Programmierers (-0.6) negative Auswirkungen auf das Projektengagement hatten (r^2=0.18). Die Untersuchung von Lakhani und Wolf kommt der vorliegenden Arbeit am nächsten. Wie Lakhani und Wolf erhebe ich das Engagement und die Motivation der Open-Source-Entwickler und untersuche mit Hilfe einer Regressionsanalyse, wie sich die Motivation auf das Engagement auswirkt. Im Gegensatz zu Lakhani und Wolf konzentriere ich mich aber ausschliesslich auf das Motiv „Spass" und erhebe neben dem Zeiteinsatz zusätzlich die Bereitschaft für zukünftiges Engagement (wie dies beispielsweise Hertel u. a. (2003) gemacht haben). Als weiterer grosser Unterschied führe ich meine Untersuchung gleichzeitig auch unter kommerziellen Software-Entwicklern durch, was mir die Möglichkeit gibt, die wesentlichen Unterschiede zwischen diesen beiden Entwicklungsmodellen und deren Auswirkungen auf das Engagement zu untersuchen.

Hemetsberger benutzte in ihrer Studie (Hemetsberger, 2003) qualitative Methoden, um die Teilnahmemotivation von Open-Source-Entwicklern zu analysieren. Auf einer Community-Site lud sie Open-Source-Entwickler ein, frei über ihr Open-Source-Engagement, über die Motive dazu, über den Nutzen und die Art ihres Engagements zu berichten. Dieser Einladung kamen 1486 Personen nach. 1139 Antworten wurden mit einer Inhaltsanalyse ausgewertet. Diese Auswertung ergab,

dass 71% der Open-Source-Entwickler angaben, durch Motive bestimmt zu sein, welche mit der Aufgabe oder der Tätigkeit zu tun haben. Darunter befanden sich 39% intrinsisch (Spass, Herausforderung etc.) und 48% extrinsisch (Gebrauchswert der Software etc.) motivierte Entwickler. 48% der Antworten erwähnten unterschiedliche Arten internalisierter Gruppenwerte (moralischer Druck, Altruismus etc.), 36% führten langfristige, utilitaristische Ziele und sozialer Austausch (Wissenserwerb, erwartete Reziprozität, Reputation) als Teilnahmegrund an. Rund 14% der Motive waren Produkt-bezogen (Qualität, Kosten etc.) und 11% betrafen sozioemotionale Beziehungen (Einsatz für Open Source allgemein, Gruppenzusammenhang etc.).

Spaeth analysierte in seiner Dissertation (Spaeth, 2005) den Quellcode von 29 Open-Source-Projekten, welcher von insgesamt 1550 Entwicklern beigesteuert worden ist, mit Methoden der sozialen Netzwerkanalyse. Das Ziel dieser Studie war, ein Verständnis über die Organisation von Open-Source-Projekten zu entwickeln sowie der Frage nachzugehen, wie sich die Entwickler innerhalb von solchen Projekten koordinieren. Der Autor konnte mit seiner Untersuchung zeigen, dass in Open-Source-Projekten allgemein ein grosser Teil der Arbeit durch einige wenige Entwickler beigesteuert wurde. Der Grad der Verknüpfung der verschiedenen Entwickler im Projekt (*inclusiveness*) variierte stark und ohne erkennbares Muster, ebenso die Zentralisierung, d.h. die Dominanz einer einzelnen Person innerhalb eines Projekts. Weiter ging die Studie der Frage nach, ob Code-Eigentum (*code ownership*) in Open-Source-Projekten existiert. Zu diesem Zweck analysierte der Autor, wieviele verschiedene Programmierer je am gleichen Programmfile Modifikationen anbrachten. Wie die Studie zeigte, wurde im Durchschnitt ein File nur von 2.4 Programmierern verändert. Dieser Durchschnitts-Wert ist im Vergleich der verschiedenen Projekte erstaunlich stabil, unabhängig von der Grösse und dem Alter der Projekte.

Tabelle 3.3: Empirische Untersuchungen zu Open Source

Studie	**Thema**	**Methode**	**Sample-Grösse**
Robles u. a. (2001)	Merkmale von Open-Source-Entwicklern	Online-Fragebogen	5’478
Jorgensen (2001)	Wirkung des Open-Source-Entwicklungs-modells auf die Motivation	Online-Fragebogen und Einzelinterviews	72
Hars und Ou (2001)	Motivation von Open-Source-Entwicklern	Online-Fragebogen	81
Dempsey u. a. (2002)	Merkmale von Linux-Entwicklern	Analyse von Linux Software Maps (LSM)	4’633
Ghosh u. a. (2002)	Merkmale von Open-Source-Entwicklern	Online-Fragebogen	2’784

Tabelle 3.3: Empirische Untersuchungen (Fortsetzung)

Studie	**Thema**	**Methode**	**Sample-Grösse**
Krishnamurthy (2002)	Merkmale von Open-Source-Projekten	Analyse der 100 aktivsten SourceForge Projekte	100
Bonaccorsi und Rossi (2003)	Merkmale von Firmen im Open-Source-Bereich	Fragebogen	146
Hertel u. a. (2003)	Motivation unter Linux Kernel-Entwicklern	Online-Fragebogen	141
Healy und Schussman (2003)	Merkmale von Open-Source-Projekten	Analyse von SourceForge Projekten	46'356
Lakhani und Wolf (2003)	Motivation von Open-Source-Entwicklern	Online-Fragebogen	648
Hemetsberger (2003)	Motivation von Open-Source-Entwicklern	Online-Fragebogen, Inhaltsanalyse	1139
Spaeth (2005)	Koordination innerhalb Open-Source-Projekten	soziale Netzwerkanalyse des Quellcodes	29

Kapitel 4

Das Flow-Konzept

Das Ziel meiner Forschung ist es, die Bedeutung von Spass für das Open-Source-Engagement zu quantifizieren. Damit ich dies erreichen kann, muss ich Spass auf zuverlässige und gültige Art operationalisieren können. In diesem Kapitel stelle ich dar, warum ich das Flow-Konzept verwende, um in meiner empirischen Untersuchung Spass zu operationalisieren.

Im ersten Unterkapitel beschreibe ich das Flow-Konzept und dessen Zusammenhang mit Spass. Im darauf folgenden Unterkapitel lege ich dar, warum das Erleben von Flow speziell bei Tätigkeiten am Computer gut beobachtet werden kann. Am Schluss dieses Kapitels erläutere ich, wie Flow empirisch gemessen werden kann.

4.1 Spass und Flow

Flow ist ein Erlebnis, das in sich selbst und unmittelbar belohnt. Ein solches Erlebnis wird *autotelisch* genannt. Von einem autotelischen Erlebnis kann gesagt werden, dass es in jedem Fall Spass macht. Bei Flow handelt es sich dementsprechend um eine Form von Spass. Hingegen trifft die umgekehrte Aussage nicht zu. Nicht jede Form von Spass ist mit dem Erleben von Flow verbunden. Beispielsweise können die Beteiligten an einem sozialen Austausch (z.B. beim geselligen Beisammensein oder bei einem Telefongespräch) Spass haben, ohne dass sie dabei Flow erleben. Spass stellt somit einen Oberbegriff dar, Flow bildet eine Unterkategorie von Spass.

Das Flow-Konzept wurde vom amerikanischen Psychologen Csikszentmihalyi entwickelt. Dieser stellte das Flow-Konzept erstmals 1975 in seiner Arbeit „Beyond Boredom and Anxiety“ vor (Csikszentmihalyi, 1975). Mit dem Flow-Konzept integrierte Csikszentmihalyi die psychologischen Arbeiten über Selbstverwirklichung und intensive Sinnerlebnisse (z.B. Maslow), über intrinsische Motivation (z.B. Berlyne, De Charms) sowie die ethologischen und ethnologischen Arbeiten zu Spiel (z.B. Callois, Huizinga). Damit kann das Flow-Konzept den Inhaltstheorien zugerechnet werden, welche einen Teilbereich innerhalb der individualpsychologischen Ansätze der Motivationstheorien darstellen. Gemeinsam mit anderen Inhaltstheorien (z.B. Bedürfnispyramide von Maslow oder Zwei-Faktoren -Theorie von Herzberg) hat das Flow-Konzept den Fokus auf die konkreten Bedürf-

nisse und Motive (z.B. Lohn, Anerkennung), die das Verhalten der Menschen steuern und bestimmen.

Aus einer psychologischen Perspektive gesehen ist Flow ein Phänomen, welches in einem Bereich angesiedelt ist, der durch die Dimensionen „Erregung“ und „Herausforderung“ aufgespannt wird und durch die Positionen „Angst“ und „Langeweile“ begrenzt ist. Aus der soziologischen Perspektive betrachtet ist Flow eine Folge von wahrgenommener Kompetenz, d.h. ein Wert im Sicherheits-Unsicherheits-Kontinuum, welches auf der einen Seite durch Entfremdung, auf der anderen Seite durch Anomie (vollkommene Regellosigkeit) begrenzt ist.

Gemäss den Untersuchungen Csikszentmihalyis machen die in Tabelle 4.1 zusammengestellten Komponenten ein Flow-Erlebnis aus.

- Handlungsanforderungen und Rückmeldungen werden als klar und interpretationsfrei erlebt, so dass man jederzeit und ohne nachzudenken weiss, was jetzt als richtig zu tun ist.
- Man fühlt sich optimal beansprucht und hat trotz hoher Anforderungen das sichere Gefühl, das Geschehen noch unter Kontrolle zu haben.
- Der Handlungsablauf wird als glatt erlebt. Ein Schritt geht flüssig in den nächsten über, als liefe das Geschehen gleitend wie aus einer inneren Logik. (Aus dieser Komponente rührt wohl die Bezeichnung „Flow“.)
- Man muss sich nicht willentlich konzentrieren, vielmehr kommt die Konzentration wie von selbst, ganz so wie die Atmung. Es kommt zur Ausblendung aller Kognitionen, die nicht unmittelbar auf die jetzige Ausführungsregulation gerichtet sind.
- Das Zeiterleben ist stark beeinträchtigt; man vergisst die Zeit und weiss nicht, wie lange man schon dabei ist. Stunden vergehen wie Minuten.
- Man erlebt sich selbst nicht mehr abgehoben von der Tätigkeit, man geht vielmehr gänzlich in der eigenen Aktivität auf (sog. „Verschmelzung“ von Selbst und Tätigkeit). Es kommt zum Verlust von Reflexivität und Selbstbewusstheit.

Tabelle 4.1: Komponenten von Flow (aus Rheinberg, 1997, S. 143)

Das Flow-Erlebnis wird begünstigt, wenn eine Person klare Ziele hat sowie eine Vorstellung, wie diese Ziele erreicht werden können und welche Regeln zu diesem Zweck befolgt werden müssen. Weiter erleichtern klare Rückmeldungen ein Flow-Erlebnis. Primäre Voraussetzungen für ein Flow-Erlebnis sind sind ein limitiertes Stimulusfeld und ein Gleichgewicht zwischen den wahrgenommenen Anforderungen und Fähigkeiten (siehe Tabelle 4.2).

1. Die Aufmerksamkeit muss auf ein limitiertes Stimulusfeld fokussiert werden können.
2. Ein Gleichgewicht zwischen den wahrgenommenen Anforderung einer Tätigkeit und dem Können der Person, wobei allerdings gelten muss, dass sowohl die Anforderungen und Fähigkeiten von der Person aus gesehen überdurchschnittlich sind.

Tabelle 4.2: Bedingungen für das Erleben von Flow gemäss Hoffman und Novak (1996)

Neben den in Tabelle 4.1 aufgeführten Komponenten weist das Flow-Erleben eine ganze Reihe weiterer Facetten auf: Der Flow-Zustand wird als in sich belohnend erfahren[1] und ist durch ein erhöhtes Mass an Verspieltheit[2] und Selbstkontrolle[3] geprägt. Weiter verstärkt der Flow-Zustand das Lernen und trägt zur positiven Selbsterfahrung[4] bei.

Diese Darstellung zeigt, dass es sich bei Flow, zumindest beim gegenwärtigen Stand der wissenschaftlichen Untersuchung, nicht um ein scharf definiertes Phänomen handelt. Trotz dieser begrifflichen Unschärfe kann das Flow-Phänomen durchaus von anderen, ähnlichen Erscheinungen abgegrenzt werden. So weisen Privette und Bundrick (1987) auf die Unterschiede zwischen Flow und den verwandten Phänomenen „Spitzen-Leistung“ (*peak performance*) und „Spitzen-Erlebnis“ (*peak experience*) hin. Flow hat mit diesen Phänomenen das Mass an Einbezogenheit (Immersion) und Aufmerksamkeit (Fokussierung) gemeinsam. Während aber *peak performance* (z.B. eine sportliche Spitzenleistung) durch eine starke Selbstwahrnehmung charakterisiert ist und die Kennzeichen von *peak experience* (z.B. Meditation) deren transpersonelle und spirituelle Qualitäten sind, zeichnet sich das Flow-Erlebnis durch das Empfinden von Spass aus sowie dadurch, dass die Aktivitäten im Flow-Zustand stukturiert und fliessend aufeinander folgen.

In ihrer Arbeit zeigen Novak und Hoffman (1997) auf, wie auf drei unterschiedliche Arten Flow modelliert werden und damit eine gewisse Ordnung in die verwirrende Beschreibung des Flow-Phänomens gebracht werden kann. Novak und Hoffman unterscheiden konzeptuelle, kausale und Flow-Kanal-Segmentierungs-Modelle.

[1] „[T]he state in which people are so intensely involved in an activity that nothing else seems to matter; the experience itself is so enjoyable that people will do it even at great cost, for the sheer sake of doing it“ (Csikszentmihalyi, 1990).

[2] „[I]n control of our actions, masters of our own fate [...] we feel a sense of exhilaration, a deep sense of enjoyment“ (Csikszentmihalyi, 1990, S. 5).

[3] „[A] related factor is the sense of control over one's environment“ (Ghani, Supnick und Rooney, 1991, S. 231).

[4] „When both challenges and skills are high, the person is not only enjoying the moment, but is also stretching his or her capabilities with the likelihood of learning new skills and increasing self-esteem and personal complexity. This process of optimal experience has been called flow“ (Csikszentmihalyi und LeFevre, 1989).

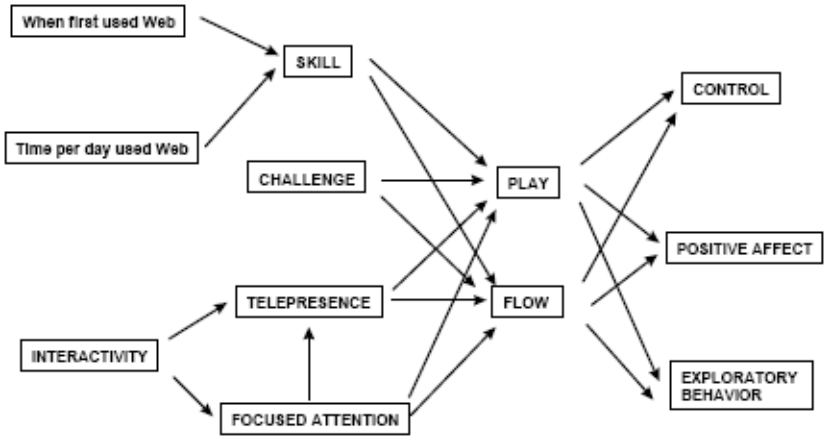


Abbildung 4.1: Konzeptuelles Modell von Novak u. a. (1997, S. 3)

Für ihre Untersuchung des Flow-Erlebens von Web-Benützern entwickelten Hoffman und Novak ein konzeptuelles Modell (siehe Abbildung 4.1), welches zwischen den Voraussetzungen, den Korrelaten und den Folge-Phänomenen des Flow-Erlebens unterscheidet. In diesem Modell steht Spiel als Korrelat zu Flow. Als Voraussetzungen sowohl für spielerisches Verhalten wie auch für Flow-Erleben setzten Hoffman und Novak die Fähigkeiten des Benützers, die Herausforderung, welche die Tätigkeit darstellt, die Telepräsenz[5] und das Fokussieren der Aufmerksamkeit auf die Tätigkeit. Telepräsenz und Aufmerksamkeits-Fokussierung haben wiederum als Voraussetzung, dass das Medium interaktiv benützt werden kann. Damit ein Benützer über die notwendigen Fähigkeiten zur Web-Interaktion verfügt, muss er diese zuerst aufgebaut haben, d.h. er muss ab einem gewissen Zeitpunkt mit einer gewissen Regelmässigkeit das Web gebrauchen. Als Folgen des spielerischen Verhaltens am Computer sowie des Flow-Empfindens haben die Web-Benützer gemäss diesem Modell das Gefühl grosser Kontrolle, sie sind positiv gestimmt und zeigen ein exploratives, neugieriges Verhalten.

Ein kausales Modell von Flow hat viele Ähnlichkeiten mit den konzeptuellen Flow-Modellen. Während aber ein konzeptuelles Modell eine Abhängigkeit bzw. Folge der einzelnen Modell-Komponenten nur postuliert, werden in einem kausalen Modell diese Beziehungen bezüglich ihrer Stärke und Richtung empirisch überprüft.

Die Flow-Kanal-Segmentierungs-Modelle (*flow channel segmentation model*) beziehen sich auf Csikszentmihalyis Postulat, dass Flow als Phänomen in einem Bereich (Kanal) zwischen Angst und Langeweile[6] angesiedelt ist. Gemäss diesem Modell kann eine Person nur dann Flow erleben, wenn die Tätigkeit einerseits genügend Herausforderung bietet und andererseits die Person die entsprechenden

[5] Unter Telepräsenz verstehen Hoffman et al. das Gefühl eines Computer-Benützers, wenn dieser vollständig in die virtuelle Welt der Computer-Interaktion versunken ist und dabei die reale Umwelt vergisst.

[6] „Beyond Boredom and Anxiety“, so der Buchtitel von Csikszentmihalyis ursprünglicher Publikation (Csikszentmihalyi, 1975).

Fähigkeiten mitbringt, die Tätigkeit zu meistern. Verfügt die Person über zu geringe Fähigkeiten, um die Herausforderung der Tätigkeit zu bewältigen, so reagiert die Person mit Überforderung und Angst. Ist umgekehrt die Herausforderung zu gering, so langweilt sich die Person an der Tätigkeit.

Auf Grund von empirischen Untersuchungen ist dieses Modell des Flow-Kanals mit der Zeit verfeinert worden. In einem ersten Schritt wurde das Modell durch einen vierten Kanal „Apathie“ ergänzt. Apathie ist gemäss diesem Vier-Kanal-Modell das Gegenstück von Flow. Apathie wird erlebt, wenn sowohl die Herausforderung wie auch die Fähigkeiten gering sind. Zu Angst kommt es bei grosser Herausforderung in Kombination mit ungenügenden Fähigkeiten. Im umgekehrten Fall wird Langeweile empfunden. Flow entsteht in diesem Modell nur bei überdurchschnittlicher Herausforderung gepaart mit ebenso grossen Fähigkeiten.[7]

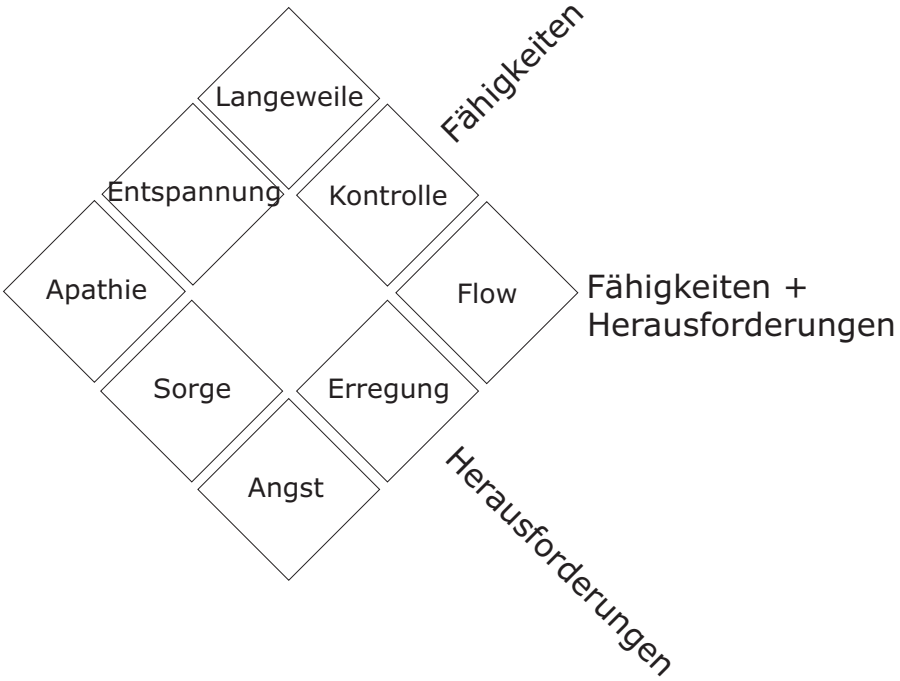


Abbildung 4.2: Acht-Kanal-Modell von Flow (aus Novak und Hoffman, 1997, S. 11)

Dieses Modell ist in der Zwischenzeit zu einem Acht-Kanal-Modell erweitert worden (siehe Abbildung 4.2), welches empirisch gut bestätigt werden konnte. In diesem Modell werden folgende zusätzliche Kanäle zwischen die bestehenden geschoben: Nehmen die Herausforderungen zu, so geht Apathie in Sorge über; bei gleichbleibenden Herausforderungen geht Apathie in Entspannung über, wenn die Fähigkeiten wachsen; Langeweile wechselt zu Kontrolle bei zunehmenden Herausforderungen, während bei gleichbleibenden Herausforderungen Angst durch Erregung abgelöst wird, wenn die Fähigkeiten zunehmen.

[7] „The flow experience begins only when challenges and skills *are above a certain level*, and are in balance“ (Csikszentmihalyi und Csikszentmihalyi, 1988, S. 260).

4.2 Flow am Computer

Die meisten Personen, die Software entwickeln, empfinden diese Tätigkeit als in hohem Mass befriedigend: „Programming [then] is fun because it gratifies creative longings built deep within us and delights sensibilities we have in common with all men“ (in Brooks, 1995, S. 8). Diese Einschätzung von Brooks wird von vielen Personen im Softwarebereich geteilt (z.B. Torvalds in Ghosh (1998)[8] oder Raymond (2000b)).

Solche Aussagen werden verständlich, wenn wir die Situation einer Person betrachten, die über Programmier-Kenntnisse verfügt und in ihrer Freizeit Software entwickelt. Für eine solche Person treffen die in Tabelle 4.2 aufgeführten Bedingungen ausgesprochen gut zu. In den allermeisten Fällen stimmt die erste Bedingung, ist doch die Interaktion mit dem Computer geprägt durch ein limitiertes und kontrolliertes Stimulusfeld.[9] Die zweite Bedingung trifft automatisch zu für selbstbestimmte Akteure, die ihre Tätigkeit frei wählen können. Für das Programmieren gilt zusätzlich, dass der Grad der Komplexität dieser Tätigkeit beliebig verändert werden kann. Geübte Computer-Programmierer können sich immer wieder neuen Herausforderungen widmen, z.B. indem sie sich mit neuen Technologien vertraut machen. In einem Umfeld, welches durch raschen technologischen Wandel geprägt ist, sind die Herausforderungen somit unerschöpflich.

Unter diesen Bedingungen ist es leicht möglich, dass ein Software-Entwickler Flow empfindet. Programmieren mit einer modernen Entwicklungsumgebung ist durch eine inkrementelle Vorgehensweise geprägt. Schritt für Schritt werden einzelne Module fertiggestellt, getestet und dem Software-Projekt als zusätzliche Funktionalität zugefügt. Nach jedem Schritt kann der Programmierer überprüfen, ob seine Eingabe fehlerfrei[10] ist und der zugefügte Code die erwartete Funktionalität ausführt.[11] Auf diese Weise ist tatsächlich gewährleistet, dass ein Schritt logisch und unterbruchsfrei in den nächsten überleitet. Mit einer vertrauten Entwicklungsumgebung sind auch die Rückmeldungen klar. Im Falle eines Fehlers kann dieser somit schnell und unkompliziert identifiziert werden.[12] All dies erlaubt dem Ent-

[8] „[M]ost of the good programmers do programming not because they expect to get paid or get adulation, but because it is fun to program. [...] The first consideration for anybody should really be whether you'd like to do it even if you got nothing at all back“ (in Ghosh, 1998, S. 9).

[9] Als Ausnahme können bestimmte Computerspiele wie z.B. *ego shooter games* gelten, wo der Takt der Stimuli durch das Spiel bestimmt ist.

[10] In modernen Entwicklungsumgebungen (den Programmen, mit welchen Software erzeugt werden kann), wird der eingegebene Code sofort kompiliert, d.h. in Maschinensprache übersetzt. Fehlerhafter Code kann nicht kompiliert werden und der Fehler im Code wird von der Entwicklungsumgebungen entsprechend markiert. Der Programmierer erhält auf diese Weise ein unmittelbares Feedback über seine Arbeit.

[11] Mit den heute üblichen Objekt-orientierten Programmiersprachen wird das sogenannte *unit-testing* bedeutend erleichtert. Ein Unit-Test ist Software-Code, welcher parallel zum eigentlichem Software-Modul erzeugt wird. Mit diesen Software-Tests wird überprüft, ob die einzelnen Funktionen des Moduls sich wie erwartet verhalten, d.h. den erwünschten Wert zurückliefern. Auf diese Weise kann sichergestellt werden, dass das erzeugte Software-Modul auch fehlerfrei im Sinne der Spezifikationen ist. In Software-Entwicklungsmethoden wie *eXtreme Programming (XP)* werden solche Tests erzeugt, noch *bevor* das eigentliche Modul implementiert wird.

[12] Moderne Entwicklungsumgebungen sind integriert, d.h. sie erlauben das Ausführen des Codes aus dem Programm-Editor heraus. Zusätzlich stellen sie einen *Debugger* zur Verfügung. Mit dem Debugger wird der Code schrittweise ausgeführt werden. In jedem Schritt kann der Entwickler den

wickler ein grosses Mass an Kontrolle über seine Tätigkeit.

Diese Umstände machen es begreiflich, dass Software-Entwickler, speziell wenn sie in ihrer Freizeit an Open-Source-Projekten arbeiten, bei ihrer Tätigkeit leicht in einen Flow-Zustand kommen und deshalb reichlich Spass an dieser Aktivität haben. Die Erläuterungen in Kapitel 4.1 geben auch Hinweise darauf, dass und warum die Möglichkeiten, Flow beim Programmieren unter kommerziellen Bedingungen zu erleben, eingeschränkt sind. Während der Arbeitszeit ist die Gefahr grösser, bei der Tätigkeit abgelenkt zu werden, wodurch die Telepräsenz gestört wird. Weiter gilt, dass unter Arbeitsbedingungen die Wahl des Software-Projekts und der Rolle darin üblicherweise nicht selbstbestimmt erfolgt. Aus diesem Grund wird die optimale Passung von Herausforderungen und Fähigkeiten seltener gefunden.

4.3 Flow messen

Flow zu messen ist eine anspruchsvolle Aufgabe. Dies liegt nicht zuletzt auch daran, dass Flow als Konstrukt nicht scharf definiert ist und von unterschiedlichen Wissenschaftlern nicht identisch interpretiert wird. Als Kritiker des Flow-Konstrukts hat Marr auf dieses Problem hingewiesen (Marr (1998) zitiert in Novak u. a. (1998, S. 2)):

> In lieu of defining and investigating a single dependent measure for flow that could be a critical aspect of the phenomenon, Csikszentmihalyi instead hypothesizes a cloud of mental events that in sum constitute flow. Flow becomes an epiphenomenon like consciousness, an emergent phenomenon that is greater than the sum of its parts. Thus Csikszentmihalyi defines flow as a 'holistic' experience that emerges from an array of simple physiological and mental events that are integrated with one another. Flow represents the voluntary investment of attention that occurs during the perception of a matching of demand and skill that produces a sense of self control, a state of joy or ecstasy, with possible outcomes including a reduction in 'ontological anxiety', a feeling of well being, and a heightened sense of awareness, playfulness, and creativity. All of these mental events were hypothesized as integral to the experience, and were further expanded through the profuse number of literary and metaphorical descriptions that were layered onto the flow experience. [...] This layer cake of interpretations has resulted in the massive obfuscation of the essential dependent variables that constitute flow.

Diese Schwierigkeiten haben Csikszentmihalyi nicht daran gehindert, nach Methoden zu suchen, mit welchen das von ihm postulierte Phänomen analysiert werden kann. Als Resultat dieser Suche nach einer Technik, mit welcher die Bedingungen und Formen von Flow ermittelt werden können, ist die *Experience Sampling*

Zustand des Systems (z.B. der Wert einer Variablen) abfragen. Auf diese Weise ist es leicht möglich zu identifizieren, an welcher Stelle des Programmflusses der Code zu einem Fehler führt.

Method (ESM) entwickelt worden (siehe Kubey und Csikszentmihalyi (1996)). Bei dieser Methode werden die Probanden mit einem Pager und einem Set von kurzen Fragebogen ausgerüstet. Im Untersuchungszeitraum (z.B. während einer Woche) wird diesen Personen in zufälligen Abständen mehrmals am Tag (z.B. 6- bis 9-mal) ein Signal zugeschickt. Die Untersuchungspersonen notieren dann bis spätestens fünf Minuten nach Eintreffen des Signals ihre aktuelle Aktivität, ihre Gemütslage und andere untersuchungsrelevante Merkmale mit Hilfe des Kurzfragebogens. Am Ende der Untersuchungsperiode werden die Fragebogen gesammelt und ausgewertet. Die ESM-Methode ist von Schallberger und Pfister (2003) verfeinert und präzisiert worden.

Mit der ESM-Methode kann reichhaltiges Material pro Person erzeugt werden. Sie ist deshalb gut geeignet, das Flow-Phänomen zu erforschen und die Bedingungen und Zusammenhänge, die das Erleben von Flow fördern oder verhindern, zu untersuchen. Die ESM-Methode ist aber sehr aufwändig. Wenn das Flow-Phänomen nicht der primäre Untersuchungsgegenstand ist, sondern wenn es in erster Linie darum geht, die Stärke des Flow-Erlebens zu messen, um damit ein anderes Phänomen erklären zu können, wenn also Flow als unabhängige Variable zur Erklärung eines anderen wissenschaftlichen Gegenstands vorkommt, dann ist die ESM-Methode weniger geeignet. In solchen Fällen müsste, damit statistisch signifikante Aussagen möglich sind, der Flow-Zustand bei einer grossen Anzahl von Personen erhoben werden. Die ESM-Methode ist aber sehr kostspielig. In einem repräsentativen Sample mit dieser Methode Flow zu messen, würde zu praktisch untragbaren Kosten führen.

Als Alternative bietet sich die Fragebogen-Methode an. Fragebogen zur Erfassung von Flow wurden in diversen Studien erfolgreich eingesetzt (z.B. Webster, Trevino und Ryan (1993), Novak und Hoffman (1997), Webster und Martocchio (1992)).

Speziell relevant für meine Untersuchung ist die Arbeit von Remy (2002). In ihrer Diplomarbeit entwickelte Remy einen Flow-Fragebogen, den sie in einem zweiten Schritt einsetzte, um das Flow-Kanal-Modell zu testen (siehe Abbildung 4.2). Der von Remy entwickelte Fragebogen enthielt 29 Items zu Flow-Bedingungen und Flow-Erleben. Weiter enthielt der Fragebogen sechs Items für eine Skala „intrinsische Motivation“ und 13 Adjektive, welche die Aktivierung und Stimmung beschrieben. Dieser Fragebogen wurde 147 Personen vorgelegt. Die Probanden wurden aufgefordert, sich an eine Situation zu erinnern, in welcher sie am Computer für eine interessante und herausfordernde Aufgabe gearbeitet hatten, und gebeten, ihre Zustimmung zu den im Fragebogen aufgeführten Aussagen auf einer 4-punktigen Skala auszudrücken. Zusätzlich wurden die Probanden gefragt, wie hoch sie den Schwierigkeitsgrad der Handlung einschätzten und wie hoch sie ihre Fähigkeiten, die Handlung zu bewältigen, beurteilten. Die Faktoren-Analysen der Antworten zeigten, dass das Flow-Erleben ein holistisches Erleben ist, welches zwar unterschiedliche Facetten aufweist, aber nicht in verschiedene Faktoren zerfällt. Die Flow-Bedingungen lassen sich auf die Faktoren „Eindeutigkeit der Aufgabe“ und „Erlebte Leichtigkeit“ zurückführen. Die Validität des Fragebogens erscheint zufriedenstellend, da alle Flow-Skalen untereinander und mit intrinsischer Motivation signifikant positiv korrelieren. Ausserdem korreliert Flow-Erleben positiv mit positiver Stimmung und Aktivation (Remy, 2002, S. 62).

Der im ersten Teil der Studie erarbeitet Flow-Fragebogen verwendete Remy in der Folge, um das Flow-Kanal-Modell zu überprüfen. Zu diesem Zweck liess Remy 63 Personen ein Computer-Spiel[13] spielen, dessen Schwierigkeitsgrad sie verändern konnte. Remy teilte die Probanden zufällig in drei Gruppen auf. Die erste Gruppe spielte das Spiel mit leichtem Schwierigkeitsgrad, die zweite Gruppe konnte den Schwierigkeitsgrad selbst bestimmen, während das Spiel bei der dritte Gruppe anfänglich auf anspruchsvollem Schwierigkeitsgrad eingestellt war und mit fortlaufender Spieldauer erhöht wurde. Auf diese Weise liess sich die Hypothese testen, dass Flow dann erlebt wird, wenn die Herausforderung optimal auf die Fähigkeiten abgestimmt sind. Remy liess die Probanden unter Laborbedingungen während 20 Minuten spielen. Nach Beendigung wurden die Teilnehmer/innen gebeten, den Flow-Fragebogen auszufüllen. Auf Grund der Antworten auf die Fragen zum Schwierigkeitsgrad der Aufgabe und zu den eigenen Fähigkeiten war es möglich, die Probanden den vier Kanälen „Langeweile“ (Fähigkeiten übersteigen Anforderungen), „Flow“ (Fähigkeiten stimmen mit den Anforderungen überein und sind beide hoch), „Apathie“ (Fähigkeiten stimmen mit den Anforderungen überein und sind beide niedrig) und „Angst“ (Anforderungen übersteigen Fähigkeiten) zuzuordnen. Gemäss Hypothese sollten am meisten Personen aus der mittleren Gruppe (welche den Schwierigkeitsgrad selbst bestimmen konnten) in der Flow-Kanal-Gruppe auftauchen und gleichzeitig im Fragebogen bezüglich des Flow-Erlebens signifikant höhere Werte angeben. Diese Vermutung konnte nicht bestätigt werden: Die mittlere Gruppe zeigte das Flow-Empfinden betreffend keine signifikant höheren Werte. Hingegen zeigte die Faktoren-Analyse der Antworten der zweiten Studie, was die Anzahl und Trennschärfe der ermittelten Faktoren betrifft, die gleichen Resultate wie bei der ersten Studie. Der Fragebogen von Remy ist somit ein sorgfältig getestetes Instrument, mit welchem sich das Flow-Erleben retrospektiv gut messen lässt.

[13] Im Experiment von Remy wurde Tetris gewählt.

Kapitel 5

Methodologie der Online-Umfrage

In diesem Kapitel behandle ich die Frage, mit welchem Instrument ich die Daten zur Berechnung der Variablen in meinem Modell erhebe. Wie ich im vorhergehenden Kapitel 4 ausgeführt habe, ist in meinem Fall für die Messung von Flow nur die Fragebogen-Methode möglich. Im ersten Teil dieses Kapitels beschreibe ich die Messmethode „Fragebogen“ sowie deren speziell für meine Untersuchung wichtigen Merkmale. Im zweiten Unterkapitel befasse ich mich mit den Kennzeichen einer standardisierten Umfrage, die im Internet durchgeführt wird. In diesem Unterkapitel geht es um die Frage, wie Online-Umfragen bezüglich ihrer Gültigkeit und Zuverlässigkeit im Vergleich zu klassischen Umfragen ausfallen. Im letzten Unterkapitel gehe ich auf die Bedeutung von Rücklaufquote und Rücklaufgeschwindigkeit ein. Je grösser die Rücklaufquote ist, desto repräsentativer sind die Aussagen, die mit den gewonnenen Daten gemacht werden können.

5.1 Die Fragebogen-Methode

Der Fragebogen ist ein klassisches Instrument der empirischen Sozialwissenschaften zur Erhebung von Daten. Im Wesentlichen besteht ein Fragebogen aus einer mehr oder weniger standardisierten Zusammenstellung von Fragen (den Fragebogenitems). Das Ziel eines Fragebogens ist es, durch eine direkte und systematische Befragung von Personen Daten zu erheben, mit denen die zugrunde liegenden theoretischen Konzepte und Zusammenhänge überprüft werden können. Ein Fragebogen stellt somit das zentrale Verbindungsstück zwischen Theorie und Analyse dar (Porst, 1996, S. 738).

Ein Fragebogenitem ist die grundlegende Einheit eines Fragebogens. Eine Frage kann offen oder geschlossen formuliert sein. Offene Fragen enthalten keine festen Antwortkategorien. Solche können allenfalls im Nachhinein, bei der qualitativen Auswertung, gebildet werden.

Bei geschlossenen Fragen sind die Anwortmöglichkeiten dagegen schon im Vornherein festgelegt. Bezüglich der Antwortkategorien können verschiedene Typen von geschlossenen Fragen unterschieden werden:

Identifikationstyp Frage zur Identifizierung der Person, Gruppe, Zeit, des Orts etc.

Ja-Nein-Typ Frage, die nur eine zweiwertige Antwort erlaubt.

Selektionstyp Frage mit vorgegebenen Alternativen. Der Befragte kann eine oder mehrere Antwortmöglichkeiten auswählen.

Skala-Frage Die Antwortkategorien beschreiben eine geordnete Intensität in einer bestimmten Dimension.

Wesentlich ist, dass die verschiedenen Typen von geschlossenen Fragen unterschiedliche Skalen haben und entsprechend unterschiedliche statistische Verfahren für die Auswertung erlauben. Identifikations- und Ja-Nein-Typen, meistens auch die Selektionstypen, sind nominal skaliert. Die Nominalskala (auch kategoriale Skala) stellt die niedrigste Ebene des Messens dar. Zur Auswertung erlaubt sie einzig das Bilden von Häufigkeiten. Als statistischer Kennwert erlaubt die Nominalskala den Modus (Dichtemittel). Der Zusammenhang zwischen zwei nominalskalierten Variablen kann mit Hilfe des Vierfelder-Korrelationskoeffizienten berechnet werden. Als Prüfstatistik für nominalskalierte Werte im Falle von unabhängigen Variablen eignet sich der Chi-Quadrat-Test, im Falle von abhängigen Variablen das Verfahren von McNemar.

Eine Skala-Frage ist dagegen mindestens ordinal, bei geeigneter Skalenwahl auch intervall skaliert. Die Ordinalskala erlaubt das Ordnen von Werten. Die gemessenen Phänomene müssen sich im Hinblick auf das beobachtete Merkmal gleichen, sich aber in der Ausprägung (Intensität, Grösse) dieses Merkmals unterscheiden. Die Ordinalskala enthält mehr Informationen als die Nominalskala. Bei der Bildung von Häufigkeitstabellen ist auch die Berechnung von kumulierten Häufigkeiten möglich. Als statistische Kennwerte können für ordinalskalierte Daten Median und mittlerer Quartilabstand berechnet werden. Der Zusammenhang zwischen zwei ordinalskalierten Variablen kann mit Hilfe des Spearman Korrelationskoeffizienten beschrieben werden. Als Prüfverfahren für ordinalskalierte Werte bei unabhängigen Stichproben eignet sich der Wilcoxon-White-Test, im Falle von abhängigen Stichproben der Wilcoxon-Test.

Die Intervallskala (auch metrische Skala) ist eine quantifizierenden Skala. Die Abstände zwischen den einzelnen Skalenwerten sind konstant. Neben der Identität und Ordnung weist diese Skala die Konstanz der Abstände und die Additivität auf. Dies macht es möglich, dass die Distanz zwischen zwei Werten auf der Intervallskala berechnet werden kann. Die Intervallskala enthält mehr Informationen als die Ordinalskala. Für intervallskalierte Daten ist die Berechnung von Mittelwert und Standardabweichung möglich. Voraussetzung ist allerdings, dass die Messwerte normalverteilt sind. Der Zusammenhang zwischen zwei intervallskalierten Variablen kann mit Hilfe der Masskorrelation (Pearson Korrelation) beschrieben werden. Als Prüfstatistik für intervallskalierte Werte im Falle von unabhängigen Variablen kann der t-Test und F-Test angewendet werden. Der t-Test prüft die Signifikanz von unterschiedlichen Mittelwerten, der F-Test diejenige von unterschiedlichen Streubreiten.

Wie angedeutet, sind bei Skala-Fragen unterschiedliche Formen möglich. Porst (in 1998, S. 29) führt dazu aus:

Skalen mit einer ungeraden Anzahl von Skalenpunkten und einer formalen Mittelkategorie (ungerade Skalen) stehen Skalen mit einer geraden Anzahl von Skalenpunkten ohne formale Mittelkategorie (gerade Skalen) gegenüber; die optimale Anzahl der Skalenpunkte ist ebenso von Bedeutung (und strittig) wie die Frage, ob man Skalenwerte von rechts nach links oder von links nach rechts auf- oder absteigen läßt, ob man eindimensionale (z.B. von 1 nach 7) oder zweidimensionale (z.B. von -3 bis +3) Skalen einsetzt, und anderes mehr.

In der Praxis haben sich unter dem Gesichtspunkt der Diskriminierungsfähigkeit numerische Skalen mit sieben plus/minus zwei Skalenpunkten bewährt (Cox 1980; empirisch überzeugend Stadtler 1983). Unter Auswertungsgesichtspunkten ist dabei wichtig, daß solche Skalen den Charakter intervallskalierter Variablen besitzen oder zumindest zu besitzen glaubhaft machen.

Die Frage, ob man eine Mittelkategorie zulassen will oder nicht, ist kein rein technisches Problem. Bietet man eine Mittelkategorie an, so läuft man Gefahr, daß Personen sie als Ausweichmöglichkeit nutzen, weil sie sich nicht auf die eine oder andere Seite der Skala einstufen wollen (mit Einführung der Mittelkategorie reduziert sich der Anteil der „weiß nicht"-Antworten - sofern eine solche Kategorie vorgegeben war - deutlich; vgl. Molenaar 1982). Zwingt man sie zu einer Einordnung, indem man die Mittelkategorie wegläßt, nimmt man ihnen die Chance, sich bewußt in der Mitte einzustufen, wobei diese bewußte Einstufung kein Ausweichen darzustellen braucht, sondern die tatsächliche und inhaltliche korrekte Plazierung in der Mitte der Skala.

In einem Fragebogen können Fragen auch indirekt formuliert sein. Bei solchen Fragen interessieren die Antworten nicht direkt. Stattdessen wird die Antwort als Indikator für die wahre Meinung der befragten Person interpretiert. Dies ist in Situationen von Bedeutung, in denen aus unterschiedlichen Gründen (z.B. Normen, strategisches Verhalten) die wahre Meinung des Probanden direkt nur schwer zu ermitteln ist. Beispiele für indirekte Fragen sind Assoziationsfragen und Informationsteste.

Der Fragebogen ist als Untersuchungsinstrument eingebettet in die Interviewsituation. Diese stellt den sozialen Kontext der Befragung dar. Die Interviewsituation kann strukturiert oder nicht-strukturiert sein. Bei einem nicht- bzw. wenig strukturierten Interview gibt der Interviewer gewisse Anstösse und möglicherweise Themen vor. Der Gesprächsverlauf ist aber flexibel. Die nächste Frage ergibt sich aus der vorherigen Antwort. Ziel eines solchen Interviews ist es, über die Meinungsstruktur der befragten Person Sinnzusammenhänge zu erfassen. Wenig strukturierte Interviews können nur qualitativ ausgewertet werden. Unstrukturierte, offene Interviews mit anschliessender qualitativer Auswertung sind hervorragend geeignet, die *Existenz* von neuen, unerwarteten Phänomenen aufzuzeigen. Oft werden solche Interviews eingesetzt, um einen Fragebogen vorzubereiten, beispielsweise zur Ermittlung der Antwortkategorien von geschlossenen Fragen.

Im Gegensatz dazu stützt sich eine stark strukturierte Befragung immer auf

einen Fragebogen. Ein strukturierter Fragebogen mit geschlossenen Fragen erlaubt die standardisierte Datenerhebung. Daten, die auf diese Weise gewonnen worden sind, können problemlos quantitativ ausgewertet werden. Ein standardisierter Fragebogen ist somit das geeignete Instrument, um die *Relevanz* eines Phänomens zu ergründen.

5.2 Fragebogenuntersuchungen im Internet

Bei einem Online-Fragebogen geben die Probanden ihre Antworten in ein Internet-Formular ein. Damit zeichnen sich Fragebogenuntersuchungen im Internet durch folgende Merkmale aus: Asynchronität, Alokalität, Automatisierbarkeit der Durchführung und Auswertung, Dokumentierbarkeit, Flexibilität, Objektivität und Ökonomie (Batinic, 2001, S. 12f).

Nach dem Ausfüllen des Fragebogens liegen die Antworten sofort in elektronisch verarbeitbarer Form vor. Die Daten müssen für die Auswertung nicht noch einmal in ein Auswertungssystem eingegeben werden und Tippfehler als mögliche Fehlerursache bei Fragebogenuntersuchungen sind eliminiert. Nach anfänglichen Investitionen in die technologische Umsetzung können Online-Fragebogen ohne zusätzlichen Aufwand von beliebig vielen Personen ausgefüllt werden. Aus diesen Gründen sind Online-Fragebogen bezüglich der Kosten sehr attraktiv.

Was die Interviewsitutation betrifft, so sind Online-Fragebogen mit brieflich zugeschickten Fragebogen vergleichbar. In beiden Fällen wird der Fragebogen ohne die Anwesenheit eines Interviewers ausgefüllt. Der Proband entscheidet autonom, wann und wo er die Umfrage ausfüllen will. Die Befragungssituation ist in diesem Sinn unkontrolliert und unstandardisiert. Asynchronität und Alokalität müssen sich bezüglich der Datenqualität nicht zwingend nachteilig auswirken, sondern können durchaus positive Effekte haben: Das Ausfüllen des Fragebogens findet im Kontext der befragten Person statt, was die Gültigkeit der von der Person gegebenen Informationen steigern kann. Dadurch, dass kein Interviewer vorhanden ist, kommen auch keine Interviewereffekte vor, was zu objektiveren Daten führen kann.

Was die Reliabilität von Online-Fragebogen betrifft, so ist zu vermerken, dass die Anonymität von solchen Befragungen zu grösserer Offenheit führen kann. Sie kann aber auch „Spassteilnehmer“ ermuntern, welche den Fragebogen mit sinnlosen Angaben ausfüllen. Das Versprechen einer Belohnung für das Ausfüllen der Umfrage kann „Glücksritter“ animieren den Fragebogen auszufüllen, um die Belohnung zu gewinnen, ohne dass auf die Qualität der Angaben Wert gelegt würde.

Wie Batinic in seiner Untersuchung (Batinic, 2001) aufzeigte, weisen die bisher vorliegenden systematischen Studien zur Validität und Reliabilität von Online-Befragungen methodische Fehler auf. Sie deuten aber darauf hin, dass die Anzahl sozial erwünschter Antworten bei Fragebogenuntersuchungen im Internet kleiner ist als bei face-to-face-Befragungen. Ebenfalls geringer ist der Anteil der fehlerhaften oder nicht ausgefüllten Items. Weiter kann sich der Umstand, dass Fragebogen im Internet als interessant beurteilt werden, positiv auf die Aufmerksamkeit der Probanden und somit die Validität der Daten auswirken. Es gibt somit keine Hinweise, dass die Reliabilität und Validität von Internet-basierten Fragebogenun-

tersuchungen geringer ist als diejenige von Papier-Bleistift-Befragungen.

5.3 Rücklaufquote und Rücklaufgeschwindigkeit

Generell werden sowohl bei der Rücklaufquote als auch bei der Rücklaufgeschwindigkeit hohe Werte angestrebt. Eine hohe Rücklaufquote wirkt sich positiv aus auf die Repräsentativität der Untersuchung, während ein schneller Rücklauf die Auswertung erleichtert und, speziell bei Untersuchungen zu einem aktuellen Thema, die Validität der Studie erhöht. Die Erfahrungen bei Fragebogenuntersuchungen allgemein zeigen, dass durch die Vergabe von Gratifikationen die Rücklaufquote signifikant erhöht werden kann. Der gleiche Effekt konnte festgestellt werden, wenn anstatt Gratifikationen (d.h. greifbaren Belohnungen, z.B. Geld oder Gutschein) Informationen (z.B. die Bereitstellung einer Zusammenfassung der Studien-Ergebnisse) angeboten werden. Allerdings fällt in diesem Fall der Effekt in kleinerem Mass an (Easton, Price, Telljohann und Boehm, 1997). Auch ist zu bedenken, dass bei einem solchen Anreiz diejenigen Personen speziell angesprochen werden, die von der Untersuchungsthematik betroffen sind, was zu einer entsprechenden Verzerrung der Stichprobe führen kann.

Die Bereitschaft an einer Umfrage teilzunehmen, hängt stark von der Thematik der Umfrage ab. In einer Untersuchung zur Teilnahmebereitschaft (Porst, Ranft und Ruoff, 1998) wurde das Thema „Freizeit, Sport, Hobbys“ (nach „Politik, Wirtschaft, Gesellschaft allgemein“) mit 8.6% der Nennungen an zweiter Stelle gesetzt. Allgemein zeigte die erwähnte Untersuchung, dass sozialwissenschaftliche Themen als interessant und unterhaltend bewertet werden und eine grundsätzliche Bereitschaft, solche Umfragen zu beantworten, vorhanden ist.

Porst (in Porst u. a., 1998, S. 16) beschreibt diesen Sachverhalt wie folgt:

> Noch mehr als bei anderen Datenerhebungsverfahren gilt für die postalische Befragung, dass die Thematik einer Untersuchung von zentraler Bedeutung ist für die Bereitschaft zur Teilnahme und damit auch für die Rücklaufquote. Beschäftigt sich der Fragebogen mit interessanten, aktuellen Themen, werden die Fragebögen schneller und vollständiger zurückgesandt. Bei Umfragen mit hohem Aufmerksamkeitswert (also einer interessanten oder aktuellen Thematik) waren „im Durchschnitt 77% Rücklauf zu verzeichnen [...] mit mittlerem Aufmerksamkeitswert 66% und mit sehr geringem Aufmerksamkeitswert lediglich 42%“ (Hippler 1988, S. 244).

Im gleichen Text berichtet Porst über eine Umfrage von Lamnek und Trepl, in welcher mit gezielten Mitteln die Rücklaufquote wirksam gesteigert werden konnte (Porst u. a., 1998, S. 18):

> Aber nicht alleine die Nachfaßaktion hatte ihre Wirkung gezeigt, sondern es gab eine ganze Reihe anderer Faktoren, mit deren Einsatz Lamnek und Trepl die Teilnahmemotivation erhöhen wollten. So wurde bei jedem Fragekomplex des Fragebogens die Numerierung neu

> bei eins begonnen, um die wahre Länge des Bogens etwas zu kaschieren. Die Fragen wurden leicht verständlich und knapp formuliert. So weit es ging, wurden statt offener geschlossene Fragen verwendet, auf Filterfragen wurde weitestgehend verzichtet. Die Wichtigkeit der Mitarbeit des Einzelnen für den Erfolg der Untersuchung wurde hervorgehoben. Es wurden Rückruf-Telefonnummern angegeben, unter denen die Probanden Fragen stellen und zusätzliche Auskünfte zu dieser Studie erhalten konnten. Ein offizieller Briefbogen des Instituts verlieh dem Anschreiben die entsprechende Seriosität. Frankierte Rückumschläge wurden beigelegt. Alle Teilnehmer nahmen automatisch an einem Preisausschreiben teil (sofern sie ihre Adresse angaben). Ein übriges zur relativ hohen Ausschöpfung bewirkte die Thematik der Untersuchung. Da die Teilnehmer in der Regel persönlich betroffen waren, ergab sich dadurch auch eine höhere Teilnahmebereitschaft.

Wie wirkt sich die spezielle Interviewsituation von Online-Umfragen auf Rücklaufquote und Rücklaufgeschwindigkeit aus? In seiner Studie „Fragebogenuntersuchungen im Internet“ (Batinic, 2001) untersuchte Batinic Rücklaufquote und Rücklaufgeschwindigkeit beim Einsatz von Online-Fragebogen. Für seine Untersuchung hatte er folgende Hypothesen aufgestellt:

- Hypothese 1: Probanden mit hoher Internetaktivität nehmen eher an Online-Befragungen teil.
- Hypothese 2: Probanden mit hoher Internetaffinität (Anzahl regelmässig genutzter Internet-Dienste) nehmen eher an Online-Befragungen teil.
- Hypothese 3: Probanden mit hoher Internetaktivität zeichnen sich durch höhere Rücklaufgeschwindigkeit aus.
- Hypothese 4: Bei Online-Befragungen ist die Anzahl fehlender Werte unabhängig von der Rücklaufgeschwindigkeit.
- Hypothese 5: Bei Online-Befragungen ist bezüglich Reliabilität kein Unterschied zwischen Früh- und Spätantworten festzustellen.

Zur Verifikation seiner Hypothesen wertete Batinic eine Panel-Befragung aus. Die Stichprobengrösse dieser Panel-Befragung betrug 6149 Personen, die Feldzeit 10 Tage. 4854 Probanden beantworteten die Umfrage (Rücklaufquote 78.9%), von diesen füllten 4551 Personen (93.8%) den Fragebogen vollständig aus. Mit diesem Test konnten die Hypothesen 2, 4 und 5 bestätigt werden, während die Auswertung bezüglich der Hypothesen 1 und 3 keine Aussagen erlaubte.

Was lässt sich aus diesen Ausführungen ableiten für das Design meiner Studie? Im Fall von Software-Entwicklern kann ich zweifellos sowohl von einer hohen Internetaktivität als auch von einer hohen Internetaffinität ausgehen. Aus diesem Grund wie auch auf Grund des im Bereich „Freizeit, Sport, Hobbys“ angesiedelten Themas kann ich auf eine akzeptable Rücklaufquote hoffen. Eine Nachfassaktion wäre zur Steigerung der Rücklaufquote wünschbar, ist aber zumindest

im Falle der Open-Source-Umfrage nur schwer zu realisieren. Um eine Nachfassaktion durchführen zu können, müsste ich über ein im Vorfeld bestimmtes Sample verfügen. Um eine möglichst hohe Quote an Antworten zu bekommen, habe ich allerdings eine Vorgehensweise gewählt, die auch schon von anderen empirischen Studien zu Open Source erfolgreich angewendet worden ist (vgl. Robles u. a. (2001), Ghosh u. a. (2002)): Ich habe meine Studie auf diversen Open-Source-Plattformen und Mailinglisten publiziert und darauf gehofft, dass die Nachricht über meine Studie innerhalb der Open-Source-Szene weitererzählt wird. Ich hatte also keinen konkreten Anhaltspunkt, wie gross mein Publikum sein würde.

Das Ziel meiner Datensammlung ist eine quantitative Analyse. Quantitative Auswertungen setzen standardisierte Datenerhebungsmethoden voraus. Aus diesem Grund stand der Fragebogentyp nicht mehr zur Disposition und mir blieb nur noch die Freiheit, zwischen den verschiedenen Formen von geschlossenen Fragen zu wählen. Da intervallskalierte Daten den grössten Informationsgehalt aufweisen, war mir daran gelegen, möglichst viele Frageitems in der Form von Skala-Fragen zu präsentieren. Bei der Anzahl Skalenpunkte entschied ich mich für eine sechspunktige Skala. Hinter dieser Wahl stand die Überlegung, bei unentschlossenen Teilnehmern eine Antworttendenz (ein „eher mehr“ oder „eher weniger“) zu erzwingen. Die Notwendigkeit, sich klar für eine Position entscheiden zu müssen, milderte ich aber durch die Möglichkeit, die Frage ausserhalb der Skala mit „weiss nicht“ zu beantworten.

Kapitel 6

Forschungsdesign

In diesem Kapitel beschreibe ich das Design meines Umfrageprojekts. Im ersten Unterkapitel wird der Aufbau des Fragebogens erläutert und begründet. In den anschliessenden Teilen erkläre ich die technische Umsetzung der Umfrage und schildere, mit welchen Methoden ich den Fragebogen getestet und zur Einsatzreife gebracht habe. Die abschliessenden Unterkapitel enthalten die genaueren Einzelheiten zur Durchführung der Umfrage.

6.1 Aufbau des Fragebogens

Um meine Forschungsfragen beantworten zu können, muss ich mittels eines geeigneten Instruments sowohl die erklärenden Variablen „Flow“ und „Freizeit“ wie auch die Kriteriumsvariable „Engagement in Open-Source-Projekten“, zweckmässig operationalisiert, messen können. Wie ich in Kapitel 5.2 ausgeführt und begründet habe, verwendete ich als Messinstrument für meine Untersuchung einen Online-Fragebogen. Um allfällige Unterschiede beim Empfinden von Flow in Abhängigkeit des Entwicklungsmodells (Open Source oder kommerziell) messen zu können, musste ich zwei Fragebogen entwerfen, die im Teil, in welchem das Flow-Empfinden erhoben wurde, identisch waren, in den übrigen Teilen aber spezifisch an die Entwicklungsmodelle angepasste Fragen enthielt.

Der Fragebogen für die Open-Source-Entwickler war wie folgt aufgebaut (siehe Anhang C): Der erste Block von Fragen (28 Items) enthielt Aussagen, welche das Empfinden von Flow während der Tätigkeit am Computer beschrieben (z.B. „Die Handlung erfolgt wie aus einem Guss“ oder „Ich bin voll und ganz bei der Sache“). Die befragten Personen konnten die Häufigkeit, mit welcher die entsprechende Feststellung zutrifft, auf einer sechspunktigen Likertskala angeben. Die Bedeutung der Skala reichte von „nie“ (linker Endpunkt) bis „immer“ (rechter Endpunkt). Fünf Items in diesem Block waren so formuliert, dass sie umgekehrt bewertet wurden, d.h. eine hohe Häufigkeit signalisierte geringes Flow-Empfinden (z.B. „Ich lasse mich von anderen Dingen ablenken.“). Die Fragen in diesem Block habe ich dem Fragebogen von Remy (2002) entnommen und angepasst an die spezifische Situation in meiner Umfrage formuliert.

Um einen möglichen Reihenfolge-Effekt zu minimieren, wurden die 28 Flow-Items in eine zufällige Reihenfolge gebracht. Der Fragebogen enthielt aber keinen

Test zur Bestimmung der Grösse eines solchen Effekts.

Anschliessend an den Block mit den Flow-Items enthielt der Fragebogen (in der Version für Open-Source-Entwickler) einen Block mit drei Fragen, mit welchen die Bereitschaft für zukünftiges Engagement für Open Source ermittelt wurde. Bei diesem Block liess ich mich von der Studie von Hertel u. a. (2003) inspirieren. Er diente als eine mögliche Operationalisierung des Engagements für Open Source.

In einem nächsten Block wurden die Open-Source-Entwickler befragt, wie sie die Open-Source-Projekte erleben. Dieser Block enthielt Fragen nach der Projekt-Vision, nach der Häufigkeit von Abgabeterminen und der Bedeutung der Fachkompetenz des Projekteigentümers (insgesamt vier Items). Diese Fragen zum Open-Source-Entwicklungsmodell wurden im Hinblick auf den Vergleich mit der Projekt-Situation in kommerziellen Software-Projekten gestellt.

Gefolgt wurde dieser Fragebogenteil von einer Gruppe von Fragen zur ursprünglichen Motivation, sich an einem Open-Source-Projekt zu beteiligen bzw. ein solches zu initiieren. In der Annahme, dass sich die Motive, einem bestehenden Open-Source-Projekt beizutreten und ein eigenes zu lancieren, unterscheiden, wurden zwei Sets mit sieben (Fall „Teilnahme“) bzw. acht (Fall „Lancierung“) Motiven präsentiert. Bei der Formulierung dieser Motive habe ich mich durch die Befunde aus empirischen Untersuchungen im Open-Source-Bereich (z.B. Ghosh u. a. (2002), Lakhani und Wolf (2003) oder Hemetsberger (2003)) leiten lassen. Wieder konnten die Probanden die Relevanz der vorgelegten Motive für ihre Situation in einer sechspunktigen Likertskala angeben. Diese reichte in diesem Fragebogenteil von „völlig unwichtig“ bis „sehr wichtig“. Zweck dieser Fragen war, die Open-Source-Entwickler auf Grund unterschiedlicher Motivationsmuster typisieren zu können.

Im anschliessenden Teil des Fragebogens wurden die Open-Source-Entwickler gefragt, wie sie ihre Projektarbeit organisieren. Konkret enthielt dieser Block Fragen zur Anzahl bisher erstellter Code-Patches und -Module (analog zur Untersuchung von Hertel u. a. (2003)), zur Anzahl der Stunden, die sie wöchentlich für Open-Source-Projekte aufwenden (inspiriert von den Studien von Hertel u. a. (2003) und Lakhani und Wolf (2003)), und des Anteils der Freizeit, welcher für das Open-Source-Engagement eingesetzt wird. Dieser Fragebogenteil enthielt demnach zwei weitere Operationalisierungen des Engagements für Open Source: Einerseits versuchte ich das Engagement mittels der bisherigen Code-Leistung zu messen, andererseits mittels der wöchentlich eingesetzten Zeit.

Im abschliessenden Teil des Fragebogens wurde nach demographischen Angaben über die Person gefragt (Anzahl Jahre Programmiererfahrung, Geschlecht, Nationalität etc.). Abgeschlossen wurde der Fragebogen mit der Frage, wie viele Minuten die befragte Person für das Ausfüllen des Fragebogens brauchte. Mit der Antwort auf diese Frage konnte die Kriteriumsvalidität des Fragebogens getestet werden. Die Fragebogen-Software erlaubte es mir, die effektiv benötigte Zeit für das Ausfüllen zu messen. Gibt nun eine Person auf die Frage nach dem benötigten Zeitaufwand einen Wert nahe dem effektiven Wert an, so ist das ein deutlicher Hinweis darauf, dass die Person bereit war, den Fragebogen sorgfältig auszufüllen.[1]

[1] Den Hinweis zur Messung der Kriteriumsvalidität auf diese Weise habe ich Batinic (2001) entnommen.

Der Fragebogen für die Programmierer in kommerziellen Software-Projekten enthielt nach dem einleitenden Teil zur Messung des Flow-Empfindens einen Block zu Arbeitsplatz und Arbeitgeber. Dieser Block enthielt drei Fragen zum Engagement (Operationalisierung der Kriteriums-Variablen) sowie vier Fragen über das Verhältnis zum Arbeitgeber. Mit diesen Fragen konnte die Arbeitszufriedenheit, aber auch die Herausforderung, welche der Arbeitsplatz bietet, ermittelt werden.

Anschliessend folgten zwei Fragen zur Beziehung zu Open Source: Setzt der kommerzielle Software-Entwickler quelloffene Software an seinem Arbeitsplatz ein, oder arbeitet er gar an einem solchen mit? Mit diesen Fragen wollte ich prüfen, wie weit Open-Source-Software auch die kommerzielle Software-Produktion durchdringt. Weiter wollte ich die Vermutung testen, ob der Gebrauch solcher Software oder die Mitarbeit in Open-Source-Projekten eine Auswirkung auf die Arbeitszufriedenheit hat.

Im nächsten Teil der Fragebogens ging es um die Projektarbeit unter kommerziellen Produktionsbedingungen. Wie häufig sind Abgabetermine spürbar, wie wichtig und spürbar sind Projekt-Visionen und die formale Autorität bzw. fachliche Kompetenz der Projektverantwortlichen. Mit den Antworten dieser Fragen konnte ich das kommerzielle Entwicklungsmodell mit dem Open-Source-Entwicklungsmodell vergleichen. Weiter enthielt dieser Block Fragen nach Anteil Zeit, in welcher die befragte Person als Projektleiter bzw. Software-Architekt und Programmierer arbeitet. Abgeschlossen wurde der Block mit der Frage nach der Anzahl der Checkins[2] in den letzten Arbeitstagen. Mit dieser Frage wollte ich die Produktivität des Entwicklers messen in der Hoffnung, damit über ein weiteres Mass für das Engagement des Programmierers zu verfügen.

Abgeschlossen wurde der Fragebogen für die kommerziellen Software-Entwickler mit demographischen Fragen (Anzahl Jahre Programmiererfahrung, Geschlecht, Beschäftigungsgrad) sowie mit der Frage nach dem Zeitaufwand zur Beantwortung des Fragebogens.

Insgesamt enthielten beide Fragebogen 53 Fragen. Die Open-Source-Version des Fragebogens enthielt 37 Skalen-Items.[3] Die restlichen 16 Items konnten über eine Auswahl (13 Stück) oder über eine freie Eingabe beantwortet werden. Alle Skalen-Items besassen ausserhalb der Skala einen zusätzlichen Auswahlpunkt „weiss nicht“ bzw. „don’t know“. Dieser Punkt war standardmässig aktiviert. Wurde die Frage nicht beantwortet, so wurde der Wert für den Punkt „weiss nicht“ übermittelt. Der Fragebogen wurde zuerst in deutscher Sprache entworfen und danach ins Englische übersetzt.

[2] Als *checkin* wird ein Eintrag in die Versionenverwaltung des Software-Projekts bezeichnet. Bei Software-Projekten, an welchen mehrere Entwickler beteiligt sind, muss der Code zentral gelagert und verwaltet werden. Ein typischer Arbeitsgang der Programmierers besteht darin, dass er den Code eines Moduls, das er bearbeiten will, zuerst auscheckt, d.h. in seiner Entwicklungsumgebung verfügbar macht. Dann erfolgt die eigentliche Arbeit am Code, z.B. das Beheben von Fehlern, das Anbringen von Kommentaren oder die Erweiterung mit neuer Funktionalität. Sind die Änderungen durchgeführt und getestet, so wird der veränderte Code als neue Version in die Versionenverwaltung eingecheckt. Mit diesem Schritt werden die Änderungen nun wieder für die anderen Projektbeteiligten verfügbar gemacht.

[3] Der Fragebogen für die Software-Entwickler im kommerziellen Bereich enthielt 42 Skalen-Items.

6.2 Programmierung des Fragebogens

Für mich war klar, dass der FASD-Fragebogen nicht in Papierform, sondern über das Internet ausgeliefert werden soll. Wie in Kapitel 5.2 ausgeführt, weist eine Online-Umfrage bezüglich Kosteneffizienz klare Vorteile auf. Ausserdem kann für Software-Entwickler, speziell aus dem Open-Source-Bereich, ein Internet-Zugang vorausgesetzt werden. Dementsprechend kann davon ausgegangen werden, dass durch die Wahl des Internets als technologische Basis für die Umfrage keine Person aus der Population, die mit dieser Studie untersucht werden soll, ausgeschlossen wird. Nicht gelöst sind allerdings bei dieser Vorgehensweise die Probleme der Selbstselektion und der Berechnung der Rücklaufquote (siehe Bandilla und Hauptmanns, 1999).

Die Fragebogen-Applikation für diese Online-Umfrage entwickelte ich selbst. Dabei setzte ich für die Kommunikation mit den Benützern die Java-Servlet-Technologie ein, für die Speicherung der Daten verwendete ich eine MySQL-Datenbank.[4]

Dadurch, dass ich die Fragebogen-Applikation selbst entwickelte, hatte ich in jedem Moment die volle Kontrolle über die Applikation. Dieser Umstand erwies sich hinsichtlich der Gestaltung des Fragebogen-Layouts, der Programmierung der Benützer-Interaktion und der Durchführung des Pretests als vorteilhaft.

Mit einer professionellen Gestaltung des Fragebogens im Browser-Fenster kann dem Publikum signalisiert werden, dass das Forschungsprojekt ernsthaft betrieben wird. Ein solches Signal erhöht bei den angefragten Personen den Anreiz, sich an der Umfrage zu beteiligen und den Fragebogen seriös zu beantworten. Aus diesen Gründen war mir daran gelegen, die Fragen auf kompakte und dennoch gut strukturierte Art zu präsentieren. Die angefragten Software-Entwickler sollten den Umfang des Fragebogens überblicken können und ein rasches Feedback darüber erhalten, wie viele Fragen schon beantwortet sind. Dies erreichte ich durch eine geeignete Wahl des Schrifttyps und der Schriftgrösse (siehe den Ausschnitt einer Bildschirmseite C.1 in Anhang C). Der eigentliche Fragebogen enthielt keinerlei Filterführung, sondern wurde auf einer einzigen, relativ langen Web-Seite angezeigt. Um die horizontale Orientierung zu erleichtern, waren die Frage-Items alternierend eingefärbt. Auf der linken Seite signalisierte ein vertikaler, roter Balken, welche Fragen noch nicht behandelt worden waren. Navigations-Elemente auf der rechten Seite erleichterten das Blättern auf der überlangen Fragebogen-Seite. Der Fragebogen war so gestaltet, dass er sowohl mit der Tastatur als auch mit der Maus bequem bedient werden konnte.

Die Benützer-Interaktion der Fragebogen-Applikation bestand aus drei Schritten. Im ersten Schritt wurde eine Willkommens-Seite angezeigt. Auf dieser Seite konnten die Benützer die Sprache auswählen (englisch oder deutsch) und sich in die Umfrage-Applikation einwählen. Zu diesem Zweck wurden die Benützer nach ihrem Username auf der jeweiligen Open-Source-Plattform gefragt. Mit der Eingabe des Benützernamens konnte dieser validiert werden. Mit dieser Massnahme wollte ich verhindern oder zumindest erschweren, dass ein Benützer, aus welchen

[4] Der Quellcode der Umfrage-Applikation steht unter http://developer.berlios.de/projects/fasd/ zur Verfügung.

Gründen auch immer, den Fragebogen mehrmals ausfüllt. Die angesprochenen Personen wurden aber darauf hingewiesen, dass eine anonyme Teilnahme an der Umfrage möglich ist. Auf der Willkommens-Seite war ein entsprechendes Kontrollfeld (Checkbox) angebracht. Hatte eine Person dieses Kontrollfeld angekreuzt, so unterblieb die Überprüfung des Benützernamens, der entsprechend auch nicht eingegeben werden musste.

Die Informationen der Eingaben auf der Einstiegs-Seite wurden auf dem Server verarbeitet, bevor die eigentliche Fragebogen-Seite eingeblendet wurde. Im Hinblick auf die statistische Auswertung wurde die gewählte Sprache sowie die Herkunft der Abfrage[5] gespeichert. Um einen allfälligen Missbrauch bei der Umfrage zu erkennen, merkte sich die Fragebogen-Applikation den eingegebenen Benützernamen und die IP-Adresse. Gleichzeitig hielt die Fragebogen-Applikation den genauen Zeitpunkt der Auslieferung der eigentlichen Fragebogen-Seite an den Benützer fest.

Die nächste Benützer-Interaktion erfolgte, nachdem die Programmierer den Fragebogen ausgefüllt und die Daten an den Server geschickt hatten. Alle Antworten wurden nun in der Datenbank gespeichert. Gleichzeitig wurde die Zeitdauer berechnet, welche die Person zum Ausfüllen des Fragebogens benötigt hatte. Nach dieser Verarbeitung wurde mit einer entsprechenden Rückmeldung für die Teilnahme an der Umfrage gedankt. Diese Feedback-Seite enthielt zusätzlich einen Hinweis auf die Mailingliste des FASD-Projekts, in welche sich die Open-Source-Entwickler eintragen konnten, um über die weiteren Schritte und Ergebnisse der Studie informiert zu werden.

6.3 Pretest

Ein Pretest ist ein Testlauf des Fragebogens unter möglichst realistischen Bedingungen. Mit einem solchen Pretest kann nicht nur das Befragungsinstrument im engeren Sinn, d.h. der Fragebogen als Abfolge von Frage-Items, sondern auch im weiteren Sinn, d.h. im Falle der FASD-Studie die technische Funktionsweise der Fragebogen-Applikation, unter Feldbedingungen überprüft werden.

Bezüglich des Fragebogens sollte ein Pretest Auskunft geben über:

- die Verständlichkeit der Fragen,
- Probleme des Befragten mit seiner Aufgabe,
- Interesse und Aufmerksamkeit des Befragten bei einzelnen Fragen,
- die Häufigkeitsverteilung der Antworten,
- die Reihenfolge der Fragen,
- Probleme des Interviewers,
- technische Probleme mit dem Fragebogen,

[5] Mit der Herkunfts-Information ist die Open-Source-Plattform gemeint, über welche die Software-Entwickler auf die FASD-Umfrage aufmerksam gemacht wurden.

- die Zeitdauer der Befragung

(aus Porst (1998, S. 35)).

Um diesbezüglich zu Informationen über meinen Fragebogen zu kommen, setzte ich für Pretest-Zwecke geeignete kognitive Techniken, speziell Probing-Verfahren sowie Paraphrasing und Confidence Rating, ein (siehe Prüfer und Rexroth (1996)). Beim *special comprehension probing* soll die Person, die den Pretest des Fragebogens ausfüllt, bestimmte Aspekte oder Begriffe einer Frage klären (z.B. „Was heisst das genau für Sie, wenn Sie ...“), beim *information retrieval probing* beschreiben, wie sie bei der Informationsbeschaffung zur Beantwortung der Frage vorgegangen ist (z.B. „Wie sind Sie zur Antwort x gekommen?“) und beim *category selection probing* begründen, warum sie aus einer Menge von Antwortvorgaben eine bestimmte Wahl getroffen hat (z.B. „Warum haben Sie sich bei dieser Frage für den Wert x entschieden?“). Beim *general probing* wird dem Pretester eine allgemein gehaltene Zusatzfrage z.B. zum Verständnis gestellt (z.B. „Gibt es etwas, das Sie bei dieser Frage nicht verstanden haben?“), beim *paraphrasing* wird diese Person gebeten, den Fragetext in eigenen Worten zu wiederholen (z.B. „Wie haben Sie diese Frage verstanden? Bitte geben Sie eine eigene Formulierung.“), während sie beim *confidence rating* gebeten wird, den Grad der Verlässlichkeit einer Antwort einzuschätzen (z.B. „Was würden Sie sagen: Ist Ihre Angabe sehr genau, ziemlich genau, eher ungenau oder grob geschätzt?“). Mit diesen Techniken ist es möglich, Formulierungsdefizite aufzudecken und zu testen, ob vermutete Probleme bei einzelnen Fragen tatsächlich vorhanden sind und wie sich diese auswirken.

Weil ich die Fragebogen-Applikation selbst entwickelt hatte, war es für mich einfach, auch eine Pretest-Variante des Fragebogens zu implementieren. Nicht nur wurde damit das Setting für den Pretest realitätsnah, zusätzlich konnte dieser Pretest auch sehr kostengünstig und dennoch effizient durchgeführt werden. In der Pretest-Version wurde den Test-Personen wie in der endgültigen Version zuerst eine Einstiegsseite mit allgemeinen Bemerkungen zur Studie sowie das Formular zur Eingabe eines Benützernamens und der Sprachauswahl präsentiert. Nach dem Einwählen wurde der eigentliche Fragebogen angezeigt, der von den Helferinnen und Helfern ebenfalls sorgfältig, aber nicht notwendigerweise wahrheitsgetreu ausgefüllt werden musste. Nach dem Absenden der Antworten wurde ein zusätzlicher Pretest-Fragebogen eingeblendet. Auf dieser Seite wurden die Antworten zu 24 Fragen, bei denen ich mögliche Probleme überprüfen wollte, wieder angezeigt und die Pretest-Personen gebeten, ihre Antworten bzw. ihr Verständnis der jeweiligen Frage entsprechend der oben ausgeführten kognitiven Techniken zu kommentieren. Abgeschlossen wurde dieser Pretest-Fragebogen durch vier Fragen, die sich auf den Fragebogen allgemein bezogen.[6]

Der Fragebogen der FASD-Studie wurde von 10 Personen auf diese Weise getestet, sowohl in englischer wie auch in deutscher Sprache. Auf Grund dieses Tests wurden neun Fragen umformuliert, eine Frage wurde gelöscht und eine neue Frage in die Umfrage aufgenommen. Weiter konnte auf Hinweis der Pretester das Lay-

[6] „Wie empfinden Sie die Länge des Fragebogens (gut, akzeptabel, zu lang, zu kurz)?“, „Wie beurteilen Sie die Verständlichkeit der Fragen?“, „Gibt es Fragen, die Sie als heikel empfinden, im Sinne einer möglichen Verletzung Ihrer Privatsphäre?“, „Wie beurteilen Sie die Gestaltung und Handhabung des Fragebogens?“.

out der Fragebogen-Seite verbessert werden. Zusätzlich wurde ein Fehler in der Verarbeitung der Fragebogen-Daten erkannt und eliminiert.

6.4 Auswahl der Open-Source-Plattformen

Nach dem Entwickeln und Testen der Online-Umfrage galt es, möglichst viele Open-Source-Entwickler zu finden, welche bereit waren, den Online-Fragebogen auszufüllen. Die Beispiele der WIDI- (siehe Robles u. a. (2001)) und FLOSS-Studie (siehe Ghosh u. a. (2002)) zeigen, dass mit einfachen Mitteln ein grosser Rücklauf erreicht werden kann. Beide Studien kündigten die Lancierung ihres Online-Fragebogens auf einer in der Open-Source-Community gut frequentierten Webseite an (üblicherweise slashdot.org) und konnten damit einen Schneeballeffekt in Gang setzen, indem Open-Source-Entwickler, welche die Umfrage schon ausgefüllt hatten, den Fragebogen in ihrem Umfeld weiterverbreiteten. Die WIDI-Studie konnte auf diese Weise rund 5478 Antworten generieren, die FLOSS-Umfrage wurden von 2784 Entwicklern ausgefüllt.

Eine andere Vorgehensweise wählten Lakhani und Wolf (2003) für ihren Hacker-Survey. In ihrer Studie wählten die Autoren aus den auf SourceForge gehosteten Open-Source-Projekten eine Untermenge aus, welche ihren Selektionskriterien genügte. In der Folge wurden die gemäss Projektbeschreibung aktiven Projekt-Leiter und -Entwickler identifiziert und mit personalisierten e-Mails auf die Studie aufmerksam gemacht. Der auf diese Weise angekündigte Fragebogen wurde von 648 Personen ausgefüllt.

Für die FASD-Studie wählte ich eine leicht variierte Vorgehensweise. Zur Infrastruktur-Ausstattung, welche die verschiedenen Open-Source-Plattformen den eingeschriebenen Projekten zur Verfügung stellen, gehören u.a. auch Mailinglisten. Mailinglisten sind eine einfache Möglichkeit der asynchronen Kommunikation der an einem Projekt beteiligten oder interessierten Personen. Eine an die Projekt-Mailingliste geschickte Mail wird automatisch an die eingeschriebenen Personen weitergeschickt. Diesen Mechanismus nützte ich aus, um die Open-Source-Entwickler auf SourceForge und GNU\Savannah auf die FASD-Studie aufmerksam zu machen. Im Falle von BerliOS waren die Betreiber dieser Plattform bereit, auf der Einstiegsseite mit einer entsprechenden Notiz meine Studie anzukündigen. Diese Vorgehensweise entsprach demnach dem Beispiel der WIDI- und FLOSS-Studie.

Alle drei in der Studie berücksichtigten Open-Source-Plattformen bieten in etwa die gleichen Dienstleistungen für die Open-Source-Projekte an. Bei allen geht es darum, für Open-Source-Projekte kostenlos die notwendigen Infrastruktur zur Verfügung zu stellen, damit die Projekte in geographisch beliebig verteilten Teams vorwärts getrieben und die Erzeugnisse dieser Projekte der interessierten Öffentlichkeit zur Verfügung gestellt werden können. Zu diesen Infrastrukturdiensten gehören Webseiten, damit sich die Projekte darstellen und ihre Produkte dokumentieren können, ein System zur Versionskontrolle des erzeugten Quellcodes, Mailinglisten und Diskussionsforen zur Kommunikation innerhalb des Entwicklerteams und zwischen Programmierern und Benutzern, ein System zum Verwalten der Fehler (Bug-Tracking) sowie Release-Management, damit die verschiedenen Versionen der installierbaren Software-Produkte sauber und benutzerfreund-

lich verwaltet werden können.

Trotz der vielen Gemeinsamkeiten gibt es auch charakteristische Unterschiede zwischen den drei Open-Source-Plattformen. GNU\Savannah ist, wie der Name zu erkennen gibt, eine Plattform, die von der Free Software Foundation (FSF) betrieben und damit stark vom Geist von Richard Stallmann geprägt wird. Bei BerliOS handelt es sich um eine deutschsprachige Open-Source-Plattform, die attraktiv ist für all jene Open-Source-Programmierer aus dem deutschen Sprachraum, die nicht primär in englischer Sprache kommunizieren wollen.

Mit rund einer Million eingetragenen Benutzern und etwa 100'000 eingeschriebenen Open-Source-Projekten ist SourceForge die mit Abstand grösste Open-Source-Plattform weltweit.[7] Der Zugang zu den Dienstleistungen von SourceForge ist äusserst niederschwellig. Das Anmeldeformular kann mit einem Zeitaufwand von ca. 15 Minuten ausgefüllt werden. Danach folgt eine formale Kontrolle und nach rund einem Tag erfolgt die Mitteilung, dass die Infrastruktur für das neu angemeldete Projekt bereitgestellt und freigeschaltet ist. Auf Grund der Grösse und des Bekanntheitsgrads von SourceForge war zu erwarten, dass der überwiegende Teil der Antworten der FASD-Studie von Benutzern dieser Plattform kommen wird (was tatsächlich auch geschehen ist, siehe Tabelle 7.3). Welche Auswirkungen hat dieser Sachverhalt auf die Qualität der gesammelten Daten?

Als Folge der grossen Attraktivität von SourceForge unter den Open-Source-Programmierern hat sich diese Plattform auch zu einem beliebten Untersuchungsgegenstand der Forschergemeinschaft zum Open-Source-Phänomen entwickelt (siehe u.a. Krishnamurthy (2002), Healy und Schussman (2003), Lakhani und Wolf (2003), Howison und Crowston (2004), Howison, Conklin und Crowston (2005), Weiss (2005)). Für meine Untersuchung relevant ist die Feststellung, dass ein Grossteil der Projekte, die auf SourceForge eingeschrieben sind, weder ein Reifestadium erreichen noch eine Entwicklergemeinschaft aufbauen können. Offensichtlich sind viele der angemeldeten Projekte wenig professionell aufgezogen. Es ist dementsprechend davon auszugehen, dass die klassischen Open-Source-Hacker, d.h. Software-Entwickler, die in ihrer Freizeit an ihren eigenen Open-Source-Projekten arbeiten, im SourceForge-Sample überrepräsentiert sind, während professionelle Open-Source-Entwickler in diesem Sample untervertreten sind. Tatsächlich betreiben die grossen Open-Source-Projekte, die meist auch von bedeutenden Software-Firmen unterstützt werden, ihre eigenen Plattformen (z.B. Apache, OpenOffice.org, Linux, Eclipse etc.). Weiter signalisiert ein Open-Source-Projekt seinen Erfolg, wenn es sich leisten kann, sich von SourceForge zu lösen und eine eigene Infrastruktur-Site (Web- und Download-Seite) aufzubauen.

In meiner Studie werde ich also mit einer Verzerrung zugunsten der Freizeit-Programmierer rechnen müssen. Dies gilt es zu bedenken, wenn auf Grund dieser Daten Aussagen über die relative Bedeutung von professionellen, d.h. bezahlten Entwicklern im Open-Source-Umfeld gemacht werden. Was aber die eigentliche Forschungsfrage betrifft, die Untersuchung des Motivationsfaktors „Spass", so halte ich dieses Sample für vorteilhaft. Der Motivationsfaktor „Spass" sollte bei Freizeit-Programmierern unverfälschter zum Tragen kommen und deshalb leichter zu untersuchen sein als in einem von professionellen Software-Entwicklern ge-

[7] Stand Sommer 2005.

prägten Umfeld, wo noch andere, speziell monetäre Dimensionen ins Spiel kommen.

6.5 Durchführung der Umfrage

Im Vorfeld der Lancierung der FASD-Umfrage wurden auf den Open-Source-Plattformen SourceForge und GNU\Savannah mit geeigneten Hilfsprogrammen die Mail-Adressen der auf den Plattformen eingeschriebenen Open-Source-Projekte herausgelesen. In der Folge wurde diese Liste von Mail-Adressen in eine kleine Java-Applikation eingespiesen, welche an jede der Adressen eine Mail verschickte. Mit dieser Mail wurden die Adressaten auf die FASD-Studie aufmerksam gemacht und die Mitarbeiter des jeweiligen Open-Source-Projekts eingeladen, an der Online-Umfrage teilzunehmen (siehe Anhang D). Diese Mails wurden in verschiedenen Wellen verschickt. Die erste Welle wurde am Montag, den 3. Mai 2004 gestartet. Insgesamt wurden mehrere zehntausend Mails an Open-Source-Projekte auf diese Weise versandt.

Zusätzlich wurde am 11. Mai 2004 von den Betreibern von BerliOS auf der Einstiegsseite dieser Plattform eine News mit dem Aufruf der FASD-Studie platziert. Die Webseite mit der Fragebogen-Applikation blieb bis zum 25. Juni 2005, d.h. während 54 Tagen geöffnet. In dieser Zeit wurden insgesamt 1330 Einträge in der Umfrage-Datenbank erzeugt, d.h. es wurden 1330 Fragebogen mehr oder weniger vollständig ausgefüllt.

Im Sommer 2004 fragte ich verschiedene Software-Firmen in der Schweiz an, ob sie bereit wären, an der FASD-Studie teilzunehmen. Nach intensiver Suche konnten schliesslich sechs Unternehmungen gefunden werden. Die Durchführung der FASD-Studie unter Software-Entwicklern im kommerziellen Bereich startete am 20. September 2004. Meine Kontaktpersonen in den teilnehmenden Firmen erhielten eine Mail, die sie mit einem Begleitkommentar versehen an die Software-Entwickler der Firma weiterleiten konnten. Nach rund einer Woche bekamen die Kontaktperson wieder eine Mail von mir. In dieser Mail informierte ich über die aktuelle Rücklaufquote und bat die Kontaktperson, aufbauend auf einer beigefügte Textvorlage, den Programmierern der Firma eine Erinnerungsmail zu schicken. Diese Nachfassaktion wurde wahrscheinlich nur von vier Firmen durchgeführt. Auf diese Weise konnten 114 Fragebogen von kommerziellen Software-Entwicklern gewonnen werden.

Teil II

Forschungsresultate

Kapitel 7

Open-Source-Umfrage

7.1 Deskriptive Auswertungen der Open-Source-Umfrage

Dieses Kapitel enthält die Ergebnisse der deskriptiven Auswertungen der Daten, die ich mit der Umfrage unter Open-Source-Entwicklern gewonnen habe. Diese Auswertungen sollen einen allgemeinen Eindruck über die Population der Open-Source-Entwickler vermitteln.

7.1.1 Allgemeines Antwortverhalten

Die Tabelle 7.1 zeigt die Übersicht über die Anzahl fehlender Werte pro gespeicherten Fragebogen. Für die Auswertungen in den folgenden Kapiteln wurden alle Fälle herausgefiltert, welche mehr als 25 fehlende Werte aufwiesen (41 Fälle, d.h. 3.1%). Zusätzlich wurden zwei Fälle ausgeschlossen, die bei den Fragen zum Spass während des Programmierens konstant den Wert „immer“ markiert hatten, sowie ein Fall, der durch einen unrealistisch hohen Wert[1] bei der Anzahl Wochenstunden für Open Source auffiel.

In Kapitel 6.1 habe ich einen Test der Validität der einzelnen Fragebogen beschrieben. Dieser beruht auf einem Vergleich der Antwort auf die Frage nach dem Zeitaufwand, welchen die Teilnehmer zur Beantwortung der Umfrage brauchten, und der effektiv dazu benötigten Zeit (siehe Tabelle 7.4). Im Falle meiner Umfrage stellte sich heraus, dass dieser Vergleich keine zusätzlichen Hinweise über die Validität der einzelnen Fragebogen erlaubt.

Tabelle 7.1: Fehlende Werte

Anzahl fehlend	Anzahl Fälle	Prozent	Kumulierte Prozent
0	638	48.0%	48.0%
1	150	11.3%	59.2%
2	60	4.5%	63.8%
3	43	3.2%	67.0%
4	24	1.8%	68.8%

[1] 144 Stunden pro Woche

Tabelle 7.1: Fehlende Werte (Forts.)

Anzahl fehlend	Anzahl Fälle	Prozent	Kumulierte Prozent
5	16	1.2%	70.0%
6	16	1.2%	71.2%
7	32	2.4%	73.6%
8	112	8.4%	82.0%
9	43	3.2%	85.3%
10	40	3.0%	88.3%
11	29	2.2%	90.5%
12	20	1.5%	92.0%
13	11	0.8%	92.8%
14	14	1.1%	93.8%
15	9	0.7%	94.5%
16	7	0.5%	95.0%
17	5	0.4%	95.4%
18	1	0.1%	95.5%
19	8	0.6%	96.1%
20	2	0.2%	96.2%
21	1	0.1%	96.3%
22	5	0.4%	96.7%
23	2	0.2%	96.8%
24	1	0.1%	96.9%
25	1	0.1%	97.0%
26	1	0.1%	97.1%
27	1	0.1%	97.1%
28	5	0.4%	97.5%
29	2	0.2%	97.7%
31	2	0.2%	97.8%
33	2	0.2%	98.0%
35	4	0.3%	98.3%
37	1	0.1%	98.3%
38	1	0.1%	98.4%
39	1	0.1%	98.5%
42	2	0.2%	98.6%
43	3	0.2%	98.9%
45	1	0.1%	98.9%
48	1	0.1%	99.0%
50	1	0.1%	99.1%
52	1	0.1%	99.2%
58	1	0.1%	99.2%
61	2	0.2%	99.4%
62	8	0.6%	100.0%
Total	1330	100%	

Tabelle 7.2: Sprache

	Anzahl	Prozent
de	279	21.2%
en	1036	78.8%
Total	1315	100%

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.3: Verwendete Open-Source-Plattform

		Anzahl	Prozent	Gültige Prozent
Gültig	SourceForge	1176	89.4%	95.7%
	GNU/Savannah	39	3.0%	3.2%
	BerliOS	14	1.1%	1.1%
	Total	1229	93.5%	100%
Fehlend		86	6.5%	
Total		1315	100%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Der Fragebogen wurde zu mehr als 78% in der englischen Version ausgefüllt (Tabelle 7.2). Von den drei in die Untersuchung einbezogenen Open-Source-Plattformen stammte mit fast 96% der überwiegende Teil von Entwicklern, die auf SourceForge eingeschrieben sind (Tabelle 7.3). Die Entwickler mussten im Durchschnitt 9 Minuten für das Ausfüllen des Online-Fragebogens einsetzen (Tabelle 7.4).

7.1.2 Demographische Angaben

Die Open-Source-Entwickler sind mehrheitlich zwischen 20 und 30 Jahre alt (Mittelwert: 28.7 Jahre, Median: 27 Jahre) wobei ein respektabler Anteil (rund 27%) zwischen 30 und 40 Jahre alt ist (Tabelle 7.5). Sie haben grossteils weniger als 5 Jahre Programmiererfahrung im Open-Source-Bereich (Mittelwert: 4.8 Jahre, Median: 3 Jahre), bloss 10% können auf mehr als 10 Jahre Praxis in diesem Feld

Tabelle 7.4: Antwortzeiten

	Mittelwert	Standard-Abweichung
Antwortzeit (Minuten)	8.93	4.78

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.5: Alter

		Anzahl	Prozent	Gültige Prozent
Gültig	10-19	129	9.7%	10.1%
	20-29	664	49.9%	51.8%
	30-39	351	26.4%	27.4%
	40-49	104	7.8%	8.1%
	50-59	30	2.3%	2.3%
	≥ 60	4	0.3%	0.3%
	Total	1282	96.4%	100%
Fehlend		48	3.6%	
Total		1330	100%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.6: Anzahl Jahre Programmiererfahrung

		Anzahl	Prozent	Gültige Prozent
Gültig	0 - 5	949	71.4%	73.2%
	6 - 10	224	16.8%	17.3%
	11 - 15	65	4.9%	5.0%
	16 - 20	31	2.3%	2.4%
	21 - 25	15	1.1%	1.2%
	> 25	12	0.9%	0.9%
	Total	1296	97.4%	100.0%
Fehlend		34	2.6%	
Total		1330	100.0%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

zurückgreifen (vgl. Tabelle 7.6).

Die Akteure im Open-Source-Bereich sind praktisch ausschliesslich männlich (98%, siehe Tabelle 7.7), leben zu 80% in einem Haushalt mit anderen Erwachsenen (Tabelle 7.9), 22% der Open-Source-Entwickler haben Kinder (Tabelle 7.10) und alle ausser 6% haben neben dem Programmieren noch diverse andere Freizeitbeschäftigungen (Tabelle 7.11). Interessant ist, dass zwar bezüglich des Alters kein Unterschied zwischen den Geschlechtern feststellbar ist, dass die Open-Source-Entwicklerinnen aber über hochsignifikant weniger Erfahrung verfügen (2.5 Jahre gegenüber durchschnittlich 4.8 Jahren bei den Männern, siehe 7.8). Der sehr kleine Frauenanteil im Open-Source-Bereich kann möglicherweise zumindest zum Teil damit erklärt werden, dass die Frauen dieses Aktionsfeld viel später zu entdecken beginnen als ihre männlichen Kollegen.

Open-Source-Entwickler sind mehrheitlich vollzeitbeschäftigt (57%, siehe Ta-

Tabelle 7.7: Geschlecht

		Anzahl	Prozent	Gültige Prozent
Gültig	männlich	1270	95.5%	98.1%
	weiblich	24	1.8%	1.9%
	Total	1294	97.3%	100.0%
Fehlend		36	2.7%	
Total		1330	100.0%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.8: Programmiererfahrung nach Geschlecht

	Anzahl Jahre		
	Mittelwert	Std.fehler	Anzahl
männlich	4.77	0.14	1262
weiblich	2.50	0.36	24
Total	4.78	0.14	1296

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.9: Wohngemeinschaft

		Anzahl	Prozent	Gültige Prozent
Gültig	ja	1025	77.1%	79.8%
	nein	260	19.5%	20.2%
	Total	1285	96.6%	100.0%
Fehlend		45	3.4%	
Total		1330	100.0%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.10: Open-Source-Entwickler mit Kindern

		Anzahl	Prozent	Gültige Prozent
Gültig	ja	285	21.4%	22.1%
	nein	1006	75.6%	77.9%
	Total	1291	97.1%	100.0%
Fehlend		39	2.9%	
Total		1330	100.0%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.11: Anzahl Hobbys (inklusive Programmieren)

		Anzahl	Prozent	Gültige Prozent
Gültig	1	71	5.3%	5.7%
	2	214	16.1%	17.1%
	3	388	29.2%	31.1%
	4	213	16.0%	17.1%
	5	73	5.5%	5.8%
	> 5	290	21.8%	23.2%
	Total	1249	93.9%	100.0%
Fehlend		81	6.1%	
Total		1330	100.0%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

belle 7.12). Mehr als ein Viertel ist in Ausbildung, während der Anteil der Teilzeitbeschäftigten 11% beträgt und derjenige der Arbeitslosen 3%.

Die antwortenden Open-Source-Entwickler haben mehrheitlich schon Erfahrung als Initiatoren und Leiter von Open-Source-Projekten (64%, siehe Tabelle 7.13). Dieser auf den ersten Blick überraschende Befund lässt sich mit der Tatsache erklären, dass es sich bei den auf den Plattformen registrierten Open-Source-Projekten zum grösste Teil um Ein-Personen-Projekte handelt (siehe dazu Krishnamurthy (2002) und Healy und Schussman (2003)). In diesem Fall ist der Entwickler automatisch auch der Projektleiter.

Gibt es einen Zusammenhang zwischen der Anzahl der Jahre, die eine Person schon im Open-Source-Bereich aktiv ist (Tabelle 7.6), und der wichtigsten je in einem Open-Source-Projekt eingenommenen Rolle? Nahe liegend ist die Hypothese, dass Projektverantwortliche schon länger aktiv sind als Programmierer und diese länger als Bug-Fixer etc. Ein Blick auf die Tabelle 7.14 scheint diese Hypothese nur teilweise zu bestätigen. Programmierer sind tatsächlich im Durchschnitt mehr als ein Jahr weniger lang im Open-Source-Bereich aktiv als Projektleiter, hingegen sind Bug-Fixer schon länger aktiv als Projektleiter. Allerdings ist der Unterschied zwischen Projektleiter und Programmierer hochsignifikant (Signifikanzniveau α = 1%), während der Unterschied zwischen Programmierer und Bug-Fixer statistisch nicht signifikant ist.

7.1.3 Nationalität

Die Tabelle 7.15 mit den Nationalitäten zeigt, dass der grösste Teil er Open-Source-Entwickler aus den USA kommt, knapp gefolgt von Deutschland. Wird die Anzahl der Open-Source-Entwickler pro Land mit der Bevölkerungsgrösse verglichen, so erscheinen die USA nur noch an 19. Stelle, während Deutschland auf Rang 8 platziert ist. In der Tabelle 7.16, in welcher die Länder nach ihrer Dichte an Open-Source-Entwicklern sortiert sind, erscheint die Schweiz an erster Stelle, gefolgt

Tabelle 7.12: Beschäftigungsgrad

		Anzahl	Prozent	Gültige Prozent
Gültig	100% beschäftigt	741	55.7%	57.3%
	90% - 99% beschäftigt	16	1.2%	1.2%
	80% - 89% beschäftigt	25	1.9%	1.9%
	70% - 79% beschäftigt	17	1.3%	1.3%
	60% - 69% beschäftigt	21	1.6%	1.6%
	50% - 59% beschäftigt	43	3.2%	3.3%
	40% - 49% beschäftigt	6	0.5%	0.5%
	30% - 39% beschäftigt	7	0.5%	0.5%
	20% - 29% beschäftigt	4	0.3%	0.3%
	10% - 19% beschäftigt	7	0.5%	0.5%
	Student	363	27.3%	28.1%
	Arbeitslos	43	3.2%	3.3%
	Total	1293	97.2%	100.0%
Fehlend		37	2.8%	
Total		1330	100.0%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.13: Bisher erreichte Projekt-Rolle

		Anzahl	Prozent	Gültige Prozent
Gültig	project leader	824	62.0%	63.9%
	developer	351	26.4%	27.2%
	bug fixer	45	3.4%	3.5%
	subscribed to a mailing list	45	3.4%	3.5%
	user	24	1.8%	1.9%
	Total	1289	96.9%	100.0%
Fehlend		41	3.1%	
Total		1330	100.0%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.14: Projekt-Rolle als Funktion der Erfahrung

	Anzahl Jahre		
	Mittelwert	**Std.fehler**	**Anzahl**
project leader	5.08	0.17	812
developer	4.04	0.24	343
bug fixer	5.27	0.77	44
subscribed to a mailing list	2.80	0.52	41
user	5.60	1.22	20
Gesamt	4.74	0.13	1260

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

von Neuseeland und Schweden. Die Ursache des Schweizer Spitzenplatzes in dieser Liste dürfte darin liegen, dass ich für die FASD-Studie speziell in der Schweizer Open-Source-Szene intensiv geworben hatte.

Tabelle 7.15: Nationalität

		Anzahl	**Prozent**	**Gültige Prozent**
Gültig	United States	248	18.6%	19.2%
	Germany	239	18.0%	18.5%
	France	68	5.1%	5.3%
	United Kingdom	60	4.5%	4.6%
	Canada	51	3.8%	4.0%
	Sweden	45	3.4%	3.5%
	Netherlands	44	3.3%	3.4%
	Italy	42	3.2%	3.3%
	Switzerland	40	3.0%	3.1%
	Australia	36	2.7%	2.8%
	India	34	2.6%	2.6%
	Austria	33	2.5%	2.6%
	Spain	26	2.0%	2.0%
	Brazil	24	1.8%	1.9%
	Poland	23	1.7%	1.8%
	New Zealand	21	1.6%	1.6%
	Portugal	19	1.4%	1.5%
	Belgium	18	1.4%	1.4%
	Finland	18	1.4%	1.4%
	Norway	16	1.2%	1.2%
	Russian Federation	15	1.1%	1.2%
	Denmark	12	0.9%	0.9%
	Romania	12	0.9%	0.9%
	Argentina	11	0.8%	0.9%

Tabelle 7.15: Nationalität (Forts.)

		Anzahl	Prozent	Gültige Prozent
	Czech Republic	9	0.7%	0.7%
	Hungary	7	0.5%	0.5%
	Mexico	7	0.5%	0.5%
	Ukraine	7	0.5%	0.5%
	Greece	6	0.5%	0.5%
	Turkey	6	0.5%	0.5%
	Estonia	5	0.4%	0.4%
	Philippines	5	0.4%	0.4%
	South Africa	5	0.4%	0.4%
	Rest	79	5.9%	6.1%
	Total	1291	97.1%	100.0%
Fehlend		39	2.9%	
Total		1330	100.0%	

Tabelle 7.16: Relative Häufigkeit

		Anzahl pro Mio.	Anzahl absolut
1	Switzerland	5.58	40
2	New Zealand	5.56	21
3	Sweden	5.09	45
4	Austria	4.08	33
5	Estonia	3.59	5
6	Norway	3.58	16
7	Finland	3.48	18
8	Germany	2.91	239
9	Netherlands	2.77	44
10	Denmark	2.26	12
11	Portugal	1.90	19
12	Australia	1.88	36
13	Belgium	1.76	18
14	Canada	1.66	51
15	Slovenia	1.51	3
16	France	1.15	68
17	United Kingdom	1.01	60
18	Czech Republic	0.88	9
19	United States of America	0.88	248
20	Ireland	0.79	3

7.1.4 Zeitliches Engagement

In der Umfrage wurden die Open-Source-Entwickler gefragt, wie viel Zeit sie für Open Source pro Woche im Durchschnitt investieren (Item 40). Mit der Frage 41 ermittelte ich den Anteil, welcher für dieses Engagement in der Freizeit geleistet wird. Basierend auf den Antworten zu diesen Fragen konnte ich das zeitliche Engagement der Open-Source-Entwickler quantifizieren und damit die Forschungsfrage 2 beantworten. Die Werte sind in Tabelle 7.17 abgebildet. Die genaue Berechnung dieser Werte ist in Anhang E beschrieben.

Diese Übersicht zeigt, dass Open-Source-Entwickler im Durchschnitt insgesamt 12.6 Wochenstunden an Open-Source-Projekten arbeiten (Median: 8 Wochenstunden). Dieser Wert liegt 1.5 Stunden unter dem von Lakhani und Wolf ermittelten Wert (Lakhani und Wolf, 2003, S. 10). Mit 7.3 Wochenstunden (58% der insgesamt aufgewendeten Zeit, Median: 4.8 Wochenstunden) findet dieses Engagement mehrheitlich in der Freizeit statt. Der Anteil am Open-Source-Engagement, welcher während der Arbeitszeit geleistet wird, beträgt aber immerhin schon stattliche 42% (Mittelwert: 5.2 Wochenstunden, Median: 1.5 Wochenstunden). Die Differenz im Zeiteinsatz in der Freizeit bzw. während der Arbeitszeit ist statistisch signifikant (T-Test bei einer Stichprobe, Signifikanzniveau $\alpha = 0.0$). Weiter zeigt die Tabelle, dass Open-Source-Entwickler bereit sind im Durchschnitt rund 27% ihrer Freizeit diesem Engagement zu widmen. Bei diesen Zahlen ist allerdings zu bedenken, dass in dieser Untersuchung die professionellen Open-Source-Programmierer untervertreten sein dürften. Professionelle Open-Source-Projekte können sich eine eigene Projekt-Infrastruktur leisten und sind deshalb weniger auf die Open-Source-Plattformen angewiesen, die ich in meiner Studie angeschrieben habe.

Die Tabelle 7.18 zeigt die Häufigkeitsverteilung des Open-Source-Engagements, welches in der Freizeit stattfindet. Aus dieser Tabelle kann man herauslesen, dass rund ein Drittel der Entwickler mehrheitlich bezahlt wird für ihr Open-Source-Engagement. Für die restlichen zwei Drittel der Programmierer finden die Open-Source-Aktivitäten vorwiegend in der Freizeit statt. Lakhani und Wolf ermittelten einen Anteil von 40% bezahlter Open-Source-Programmierer (Lakhani und Wolf, 2003, S. 9), wobei sie nicht angaben, wieviel des Open-Source-Engagements dieser Entwickler während der Arbeitszeit erbracht wird.

Die Häufigkeitsverteilung (siehe Abbildung 7.1) zeigt zwei Spitzen an den beiden Endpunkten. Diese Verteilung legt eine Unterscheidung in *Professionals*

Tabelle 7.17: Zeiteinsatz für Open Source (Stunden pro Woche)

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
Zeiteinsatz insgesamt	12.56	14.18	1228
Zeiteinsatz in Freizeit	7.31	8.23	1215
Zeiteinsatz während Arbeitszeit	5.22	10.82	1215
Anzahl Wochenstunden Freizeit	26.70	24.39	1210

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

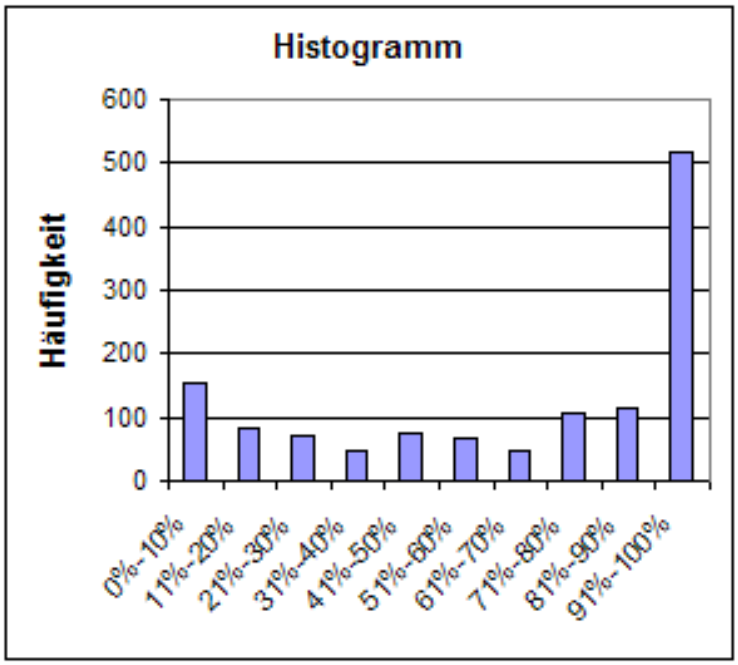


Abbildung 7.1: Anteil Zeiteinsatz in Freizeit

und *Open-Source-Hacker* nahe. Als *Professionals* bezeichne ich diejenigen Open-Source-Entwickler, die weniger als 10% des Engagements in der Freizeit erbringen, d.h. praktisch ausschliesslich während der Arbeitszeit für Open Source aktiv sind, während es sich bei den Programmierern, bei denen das Open-Source-Engagement zu mehr als 90% in der Freizeit stattfindet, zweifellos um die klassischen *Open-Source-Hacker*[2] handelt. Der Anteil der Professionals am Sample beträgt knapp 12%, während die Open-Source-Hacker rund 40% zum Sample beitragen.

Diese Interpretation kann ich mit Hilfe der Frage 33 verifizieren. In diesem Item wurde danach gefragt, wie häufig die Open-Source-Entwickler in ihren Projekten Abgabetermine verspürten. Wir erwarten, dass eine solche Frage nur für Professionals relevant ist, nicht aber für die in der Freizeit aktiven Open-Source-Hacker. Diese Erwartung wird bestätigt (siehe Tabelle 7.19). Die Open-Source-Hacker weisen für die Fragen nach Abgabeterminen einen hochsignifikant tieferen Wert auf als die Professionals ($\alpha = 0.0\%$).

Wird das zeitliche Engagement insgesamt differenziert nach diesen Typen untersucht, so stellt sich heraus, dass die Professionals mit durchschnittlich 14.6 Wo-

[2]Im deutschen Sprachgebrauch wird der Begriff *Hacker* sowohl für Software-Entwickler im Allgemeinen wie auch für Programmierer gebraucht, die Computer-Systeme und Passwörter etc. knacken. Im englischen Sprachgebrauch wird für letztere Computer-Benützer der Begriff *Cracker* verwendet und der Terminus *Hacker* ausschliesslich für im legalen Bereich arbeitende Software-Entwickler, insbesondere auch im Open-Source-Bereich, benutzt. Aus diesem Grund ist *Hacker* im deutschen Sprachraum eher negativ konnotiert, während diesem Begriff im englischen Sprachgebrauch eine positive Bedeutung zukommt. In dieser Studie verwende ich den Begriff *Hacker* im differenzierteren, englischsprachigen Sinn.

Tabelle 7.18: Anteil Zeiteinsatz in Freizeit

		Anzahl	Prozent	Gültige Prozent	Kumulierte Prozent
Gültig	0%-10%	153	11.5%	11.9%	11.9%
	11%-20%	83	6.2%	6.5%	18.4%
	21%-30%	71	5.3%	5.5%	23.9%
	31%-40%	48	3.6%	3.7%	27.6%
	41%-50%	74	5.6%	5.8%	33.4%
	51%-60%	67	5.0%	5.2%	38.6%
	61%-70%	49	3.7%	3.8%	42.4%
	71%-80%	107	8.0%	8.3%	50.7%
	81%-90%	114	8.6%	8.9%	59.6%
	91%-100%	518	38.9%	40.4%	100.0%
	Total	1284	96.5%	100%	
Fehlend		46	3.5%		
Total		1330	100%		

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.19: Häufigkeit von Abgabeterminen

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
Professional	3.04	1.55	143
Hacker	2.26	1.18	495

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.20: Zeiteinsatz der Typen

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
Professional	14.60	24.17	131
Hacker	9.69	9.54	496

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

chenstunden ein hochsignifikant grösseres Engagement zeigen als die Hacker (9.7 Wochenstunden, siehe Tabelle 7.20).

7.1.5 Flow-Erleben und Open Source

Die Tabellen in diesem Abschnitt zeigen die Mittelwerte und Standard-Abweichungen der Frage-Items zum Flow-Erleben, zur Bereitschaft für zukünftiges Engagement sowie zur Organisation von Open-Source-Projekten. Alle Items wurden durch die Wahl eines Werts auf einer sechspunktigen Skala (von 1 bis 6) beantwortet.

Bezüglich der Freude am Programmieren erhält die Aussage 22: „Die Handlung macht mir Spass“ mit einem Wert von 5.25 die höchste Zustimmung. Gleichzeitig weist dieses Item die kleinste Standard-Abweichung auf (0.75). Die niedrigste Zustimmung weist die Aussage 9: „An Vergangenes oder Zukünftiges denke ich kaum“ mit dem Mittelwert von 3.34 auf. Zugleich hat diese Aussage mit 1.46 die grösste Standard-Abweichung (Tabelle 7.21).

Bezüglich des zukünftigen Engagements weist die Aussage 29: „Ich freue mich auf weitere Entwicklungstätigkeiten für Open-Source-Software“ klar die höchste Zustimmungsrate (5.06) bei gleichzeitig kleinster Streuung (0.98) auf (Tabelle 7.22). Die Tabelle 7.23 über das Erleben der Arbeit in Open-Source-Projekten zeigt ein einheitliches Bild, was die Streuung der Werte betrifft. Wenig erstaunt der niedrige Wert von 2.57 zur Frage 33: „Bei Ihrer Arbeit für Open-Source-Projekte: Wie häufig sind Abgabetermine spürbar?“.

Befragt nach den Teilnahmemotiven erhält bei den Projektmitgliedern das Motiv, mit dem Projekt-Engagement neue Fähigkeiten erwerben zu können, die grösste Zustimmung (Tabelle 7.24). Die geringste Rolle spielt das Motiv, vom Arbeitgeber zur Projektmitarbeit beauftragt worden zu sein. Auch bei den Projektverantwortlichen spielt der Arbeitgeber bei der Lancierung eines Open-Source-Projekts die geringste Rolle. Die grösste Bedeutung für das Starten eines solchen Projekts hat der Umstand, das sich der Open-Source-Entwickler dadurch Spass verspricht, vor den beiden Motiven, eine bestimmte Funktionalität zu benötigen sowie etwas für die Open-Source-Bewegung machen zu wollen (Tabelle 7.25).

Tabelle 7.21: Flow-Erleben

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
1: I lose my sense of time.	4.16	1.26	1272
2: I cannot say how long I've been with programming.	3.81	1.44	1252
3: I am in a state of flow when I'm working.	4.58	1.07	1243
4: I forget all my worries when I'm working.	4.39	1.30	1279
5: It's easy for me to concentrate.	4.63	1.09	1278
6: I'm all wrapped up in the action.	4.66	1.00	1247
7: I am absolutely focused on what I'm programming.	4.75	1.04	1278
8: The requirements of my work are clear to me.	4.44	1.21	1270
9: I hardly think of the past or the future.	3.34	1.46	1259
10: I know exactly what is required of me.	4.10	1.21	1261
11: There are many things I would prefer doing. (-)	3.89	1.39	1255
12: I feel that I can cope well with the demands of the situation.	4.62	0.94	1255
13: My work is solely motivated by the fact that it will pay for me. (-)	4.78	1.30	1265
14: I always know exactly what I have to do.	3.85	1.18	1277
15: I'm very absent-minded. (-)	3.98	1.45	1217
16: I don't have to muse over other things.	3.43	1.25	1176
17: I know how to set about it.	4.22	1.02	1166
18: I'm completely focused.	4.50	1.01	1274
19: I feel able to handle the problem.	4.86	0.84	1276
20: I am extremely concentrated.	4.63	1.03	1273
21: I'm looking forward to my programming work.	5.01	0.91	1270
22: I enjoy my work.	5.25	0.75	1277
23: I feel the demands upon me are excessive. (-)	4.19	1.30	1256
24: Things just seem to fall into place.	4.08	1.11	1241
25: I forget everything around me.	3.95	1.29	1269
26: I accomplish my work for its own sake.	4.41	1.25	1227
27: I completely concentrate on my programming work.	4.63	1.01	1271
28: I am easily distracted by other things. (-)	4.05	1.21	1280

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.22: Zukünftiges Engagement

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
29: I'm looking forward to further development activities for open source software.	5.06	0.98	1263
30: I'm prepared to increase my future commitment in the development of open source software.	4.49	1.19	1255
31: With one more hour in the day, I would program open source software.	4.22	1.40	1233

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.23: Arbeiten in Open-Source-Projekten

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
32: How often are the open source projects you work on based on a definite project vision?	4.50	1.17	1243
33: How often is there a deadline for your open source projects?	2.57	1.34	1258
34: How important is the vision behind an open source project for you to participate in the project?	4.91	1.12	1267
35: How important is the professional competence of the project leader for your commitment in an open source project?	4.83	1.17	1246

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.24: Teilnahmemotive als Projektmitglied

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
1: It was important for my work to have a certain functionality; that's why I joined the open source project and got involved.	4.20	1.55	1198
2: My employer asked me to participate in the open source project because he needed its functionality.	1.98	1.45	1184
3: Because I wanted to do something for the open source community.	4.57	1.30	1220
4: The project promised to be fun.	4.61	1.28	1216
5: My colleagues motivated me to participate in the open source project.	2.43	1.53	1194
6: By participating in the open source project you could become famous.	2.70	1.51	1212
7: Because I wanted to learn and develop new skills.	5.00	1.18	1232

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.25: Motive als Projektverantwortlicher

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
1: I needed a certain functionality for my work, and I wanted to make this functionality available as an open source application.	4.64	1.69	1049
2: My employer asked me to start the open source project because he needed the functionality.	1.92	1.45	1026
3: My employer asked me to start the open source project because he could earn money with the application.	1.57	1.15	1022
4: Because I wanted to do something for the open source community.	4.63	1.38	1063
5: One open source project in the past didn't develop as desired, therefore I had to start my own.	3.32	1.79	1027
6: The project promised to be fun.	4.74	1.31	1052
7: I needed assistants to complete a software project.	3.10	1.71	1033
8: With an open source project you could become famous.	2.76	1.61	1048

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

7.2 Engagement, Einsatzbereitschaft und Produktivität der Open-Source-Entwickler

Im Modell 2.1, mit welchem ich das Engagement der Open-Source-Programmierer erklären will, kommt dieses Engagement als unabhängige Variable vor. Die Kriteriumsvariable *Engagement* habe ich auf drei unterschiedliche Arten operationalisiert: Als zeitliches Engagement (Zeiteinsatz in Stunden pro Woche), als Einsatzbereitschaft für zukünftiges Engagement und als Leistung (Output pro Zeiteinheit). In diesem Kapitel will ich untersuchen, wie gut sich die erhobenen Daten zur Beschreibung des Open-Source-Engagements eignen.

7.2.1 Engagement

Mit den Fragebogenitems 40 bis 42 habe ich Angaben zum zeitlichen Engagement für Open-Source-Projekte erhoben. Mit Hilfe diese Angaben sowie der Information zum Beschäftigungsgrad (Item 51) war es mir möglich, den durchschnittlichen Zeiteinsatz der befragten Personen für Open Source sowohl in der Freizeit wie auch während der Arbeitszeit zu berechnen (siehe dazu auch Anhang E und Tabelle 7.17). Welche Faktoren beeinflussen das zeitliche Engagement von Open-Source-Entwicklern?

Engagement in Abhängigkeit der Projekt-Rolle

Eine naheliegende Hypothese ist, dass sich das Engagement steigert, je wichtiger die Rolle ist. Programmierer sollten also ein grösseres Engagement zeigen als Personen, die sich als Bug-Fixer bezeichnen, und Projektleiter wieder ein grösseres verglichen mit Entwicklern, die nur als Projektmitarbeiter aktiv sind.

Mit einer einfaktoriellen Varianzanalyse kann diese Hypothese überprüft werden. Diese Analysen wurden sowohl mit den insgesamt pro Woche für Open Source eingesetzten Stunden (Frage 40) wie auch für die berechneten Werte Wochenstunden in Freizeit und während der Arbeitszeit als abhängigen Variablen berechnet. Als unabhängige Variable habe ich die Antwort auf Frage 43 zur wichtigsten bis anhin in einem Open-Source-Projekt eingenommenen Position verwendet.

Tabelle 7.26: Zeiteinsatz als Funktion der Rolle

	Stunden/Woche		
	Mittelwert	**Std.fehler**	**Anzahl**
project leader	13.97	0.51	795
developer	11.1	0.80	335
bug fixer	5.64	1.06	40
subscribed to a mailing list	5.72	1.22	33
user	4.93	1.34	14
Gesamt	12.58	0.41	1217

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

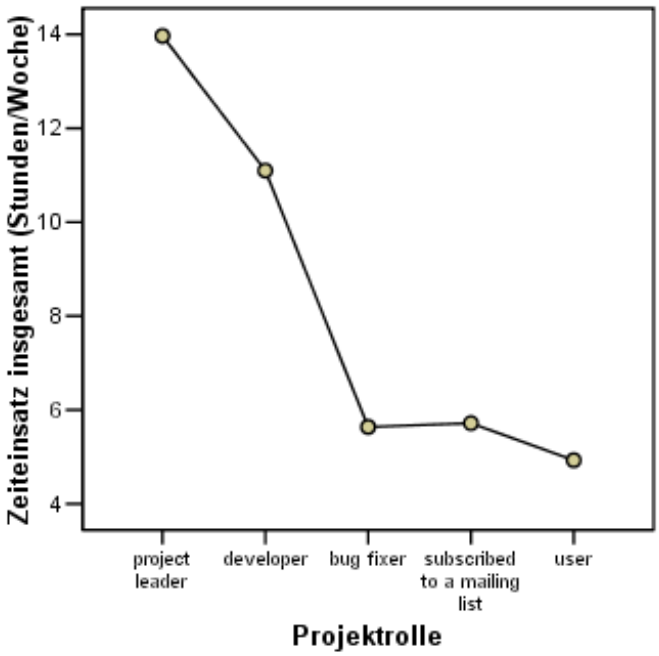


Abbildung 7.2: Zeiteinsatz als Funktion der Rolle

Die Tabelle 7.26 zeigt eine stetige Zunahme des zeitlichen Engagements, je wichtiger die Rolle ist, die der Programmierer in einem Open-Source-Projekt einnimmt bzw. eingenommen hat (siehe auch Abbildung 7.2). Ein F-Test bestätigt, dass die Mittelwerte signifikant verschieden sind (Signifikanzniveau $\alpha = 1\%$), allerdings nur für den Unterschied zwischen Projektleiter und Programmierer. Mit der Rolle des Open-Source-Entwicklers können rund 2% der Varianz im zeitlichen Engagement erklärt werden.

Wird Open Source in der Freizeit programmiert, so wiederholt sich der Zusammenhang zwischen Projekt-Rolle und zeitlichem Engagement (siehe Tabelle 7.27 und Abbildung 7.3). Wieder nimmt der Zeitaufwand, den ein Entwickler in ein Open-Source-Projekt investiert, zu, je wichtiger seine Rolle im aktuellen Projekt ist oder in den bisherigen Projekten war. Wieder ist der Unterschied zwischen Projektleiter und Programmierer statistisch signifikant (Signifikanzniveau $\alpha = 1\%$).

Wird das zeitliche Engagement von Open-Source-Entwicklern während der Arbeitszeit untersucht, so verwischen sich die Unterschiede, wie Tabelle 7.28 und Abbildung 7.4 zeigen. Der Unterschied zwischen einem Projektleiter und einem Programmierer ist nun nicht mehr statistisch signifikant.

Mit diesen Resultaten ergibt sich folgendes Bild: Findet das Engagement für Open Source in der Freizeit statt, so korreliert Engagement mit Projektverantwortung: höheres Engagement wirkt sich entsprechend in der Position aus. Findet das Engagement aber während der Arbeitszeit statt, ist dieses also in irgendeiner Form bezahlt, so ist dieses Muster nicht mehr gültig. Ein Programmierer kann auf bezahl-

Tabelle 7.27: Zeiteinsatz in Freizeit als Funktion der Rolle

	Stunden/Woche		
	Mittelwert	**Std.fehler**	**Anzahl**
project leader	8.39	0.31	790
developer	5.87	0.39	329
bug fixer	3.62	0.79	40
subscribed to a mailing list	3.07	0.55	33
user	2.13	0.72	14
Gesamt	7.32	0.24	1206

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

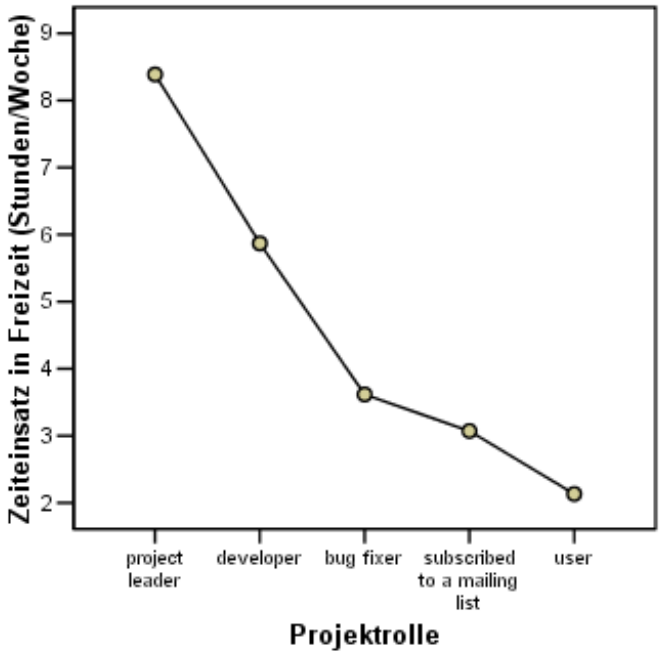


Abbildung 7.3: Zeiteinsatz in Freizeit als Funktion der Rolle

Tabelle 7.28: Zeiteinsatz während Arbeitszeit als Funktion der Rolle

	Stunden/Woche		
	Mittelwert	**Std.fehler**	**Anzahl**
project leader	5.57	0.38	790
developer	5.18	0.66	329
bug fixer	2.02	0.50	40
subscribed to a mailing list	2.65	0.94	33
user	2.80	1.08	14
Gesamt	5.24	0.31	1206

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

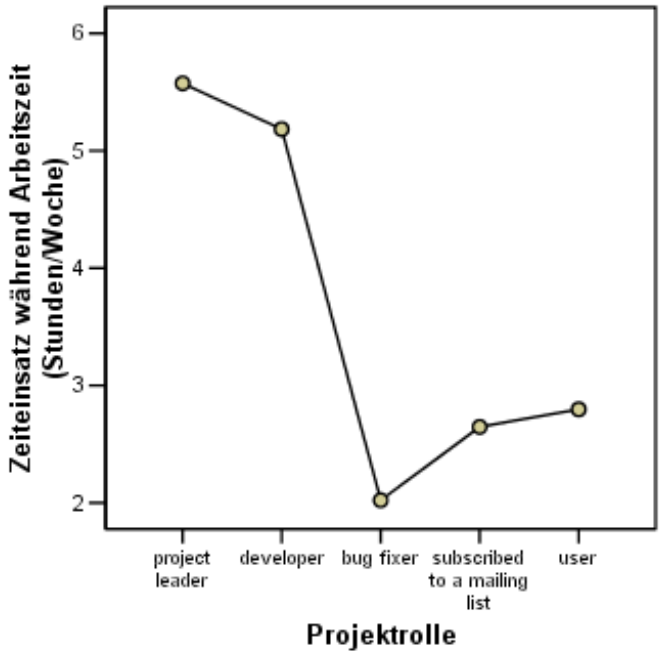


Abbildung 7.4: Zeiteinsatz während Arbeitszeit als Funktion der Rolle

Tabelle 7.29: Zeiteinsatz in Freizeit als Funktion der Anzahl Hobbys

	Stunden/Woche		
	Mittelwert	**Std.fehler**	**Anzahl**
1	7.27	1.04	67
2	8.17	0.68	202
3	7.15	0.40	365
4	7.07	0.48	201
5	7.65	1.00	70
> 5	6.71	0.51	268
Gesamt	7.25	0.24	1173

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

ter Basis zeitlich viel investieren, ohne dass er deswegen im Projekt eine wichtigere Rolle einnimmt.

Engagement in Abhängigkeit der Anzahl der Hobbys

Wird das zeitliche Engagement für Open Source (in der Freizeit) in Abhängigkeit der Anzahl der Hobbys des Software-Entwicklers untersucht, so zeigt sich ein maximales Engagement, falls der Programmierer neben Open Source eine zusätzliche Freizeitbeschäftigung hat. Zusätzliche Hobbys gehen tendenziell auf Kosten des Open-Source-Engagements (siehe Abbildung 7.5 und Tabelle 7.29). Allerdings ist der Unterschied zwischen den Werten nicht signifikant.

Engagement und Beschäftigungsgrad

Wird das zeitliche Engagement für Open Source in der Freizeit in Abhängigkeit des Beschäftigungsgrads des Programmierers untersucht, so zeigt die Abbildung 7.6 die erwartete Tendenz: Ein Entwickler programmiert in der Freizeit umso mehr, je geringer sein Beschäftigungsgrad ist. Der Unterschied zwischen einer Person, die zu 100% beschäftigt ist (rund 6.3 Stunden/Woche, siehe Tabelle 7.30), und einem Studenten (8.1 Stunden/Woche) ist statistisch hochsignifikant (Signifikanzniveau α = 1%), ebenso der Unterschied zwischen Studenten und Arbeitslosen. Nicht signifikant ist hingegen der Unterschied zwischen einer zu 100% und der zu 70% beschäftigten Person, da letztere Gruppe nur 15 Personen umfasst.

Weitere Abhängigkeiten

Weitere Abhängigkeiten des zeitlichen Engagements konnten nicht festgestellt werden. So ist der wöchentliche Zeiteinsatz für Open Source unabhängig vom Alter und der Erfahrung des Open-Source-Entwicklers.

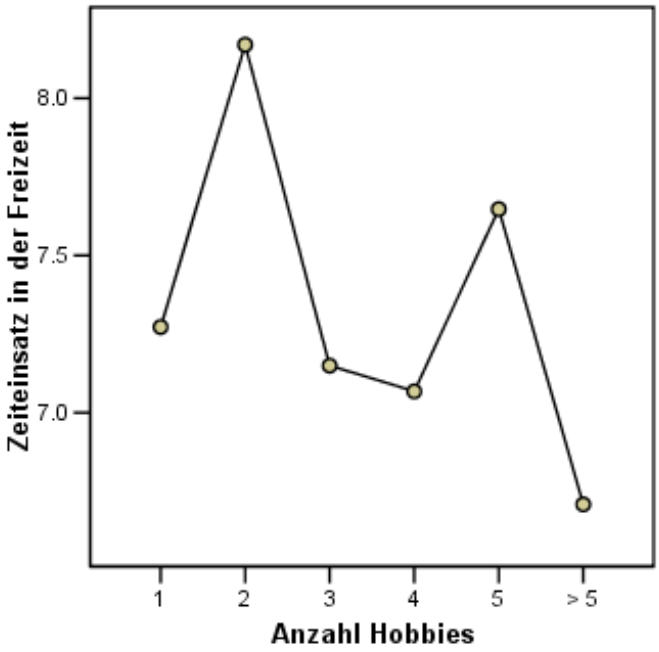


Abbildung 7.5: Zeiteinsatz in Freizeit als Funktion der Anzahl Hobbys

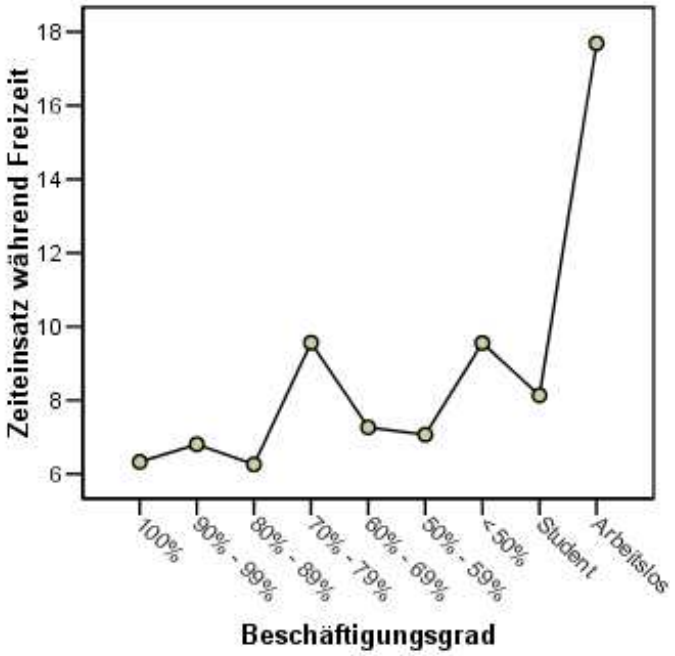


Abbildung 7.6: Zeiteinsatz in Freizeit als Funktion des Beschäftigungsgrads

Tabelle 7.30: Zeiteinsatz in Freizeit als Funktion des Beschäftigungsgrads

	Stunden/Woche		
	Mittelwert	**Std.fehler**	**Anzahl**
100%	6.33	0.25	701
90% - 99%	6.81	1.66	16
80% - 89%	6.26	1.58	23
70% - 79%	9.56	1.08	15
60% - 69%	7.27	1.54	21
50% - 59%	7.07	1.00	38
$< 50\%$	9.56	2.29	18
Student	8.13	0.48	338
Arbeitslos	17.69	3.20	37
Gesamt	7.32	0.24	1207

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

7.2.2 Einsatzbereitschaft

Mit den Fragebogenitems 29 bis 31 habe ich Angaben über die Bereitschaft für zukünftiges Engagement erhoben (siehe Tabelle 7.22). Können diese drei Items auf einen zugrunde liegenden Faktor *Einsatzbereitschaft* reduziert werden? Eine Faktorenanalyse mit diesen drei Variablen zeigt tatsächlich, dass ein einziger Faktor (mit einem Eigenwert von mehr als 2)[3] rund 67% der Varianz der einzelnen Items erklärt, während die Kommunalitäten der rotierten Lösung von 59% (Item 31) über 69% (Item 29) bis zu 74% (Item 30) reichen. Auch der Wert für Cronbachs α für diesen Faktor ist mit 0.74 zufrieden stellend.[4] In der weiteren Untersuchung werde ich deshalb auch mit einem aus dem Durchschnitt dieser drei Items gebildeten Gesamtindex *Einsatzbereitschaft* rechnen. Welche Faktoren bestimmen diese Einsatzbereitschaft?

Einsatzbereitschaft in Abhängigkeit der Projekt-Rolle

Wie ändert sich die Bereitschaft für zukünftiges Engagement in Abhängigkeit von der Projekt-Rolle? Entgegen den Erwartungen sind es nicht die Projektleiter, welche die grösste Einsatzbereitschaft zeigen, sondern die am Projekt beteiligten Programmierer (Tabelle 7.31). Am ausgeprägtesten ist der Unterschied im Item 30 (Signifikanzniveau α = 1%, siehe Abbildung 7.7), während er beim Gesamtindex nur schwach signifikant ist (α = 10%). Dies kann so gedeutet werden, dass Entwickler noch „hungriger“ nach Open Source sind als Projektleiter und somit ein grösseres Potential verspüren, ihr Engagement zu erweitern.

[3] Auch der Screeplot dieser Faktorenanalyse legt eine Reduktion auf bloss einen Faktor nahe.
[4] Cronbachs α wird häufig als Mass für die Reliabilität einer Skala verwendet.

Tabelle 7.31: Einsatzbereitschaft als Funktion der Projekt-Rolle

	Skalenwert		
	Mittelwert	**Std.fehler**	**Anzahl**
29: Ich freue mich auf weitere Entwicklungstätigkeiten für Open-Source-Software.			
project leader	5.10	0.03	796
developer	5.10	0.05	339
bug fixer	4.93	0.14	41
subscribed to a mailing list	4.60	0.20	43
user	4.19	0.34	21
Gesamt	5.06	0.03	1240
30: Ich bin bereit, mein zeitliches Engagement bei der Entwicklung von Open-Source-Software in Zukunft zu vergrössern.			
project leader	4.44	0.04	792
developer	4.66	0.06	336
bug fixer	4.51	0.14	41
subscribed to a mailing list	4.11	0.19	44
user	4.05	0.31	21
Gesamt	4.49	0.03	1234
31: Eine zusätzliche Stunde Zeit im Tag würde ich zum Programmieren von Open Source verwenden.			
project leader	4.22	0.05	779
developer	4.32	0.07	330
bug fixer	4.03	0.19	40
subscribed to a mailing list	4.02	0.23	41
user	3.38	0.35	21
Gesamt	4.22	0.04	1211
Gesamtindex Einsatzbereitschaft			
project leader	4.59	0.03	803
developer	4.70	0.05	340
bug fixer	4.47	0.14	42
subscribed to a mailing list	4.27	0.16	44
user	3.87	0.30	21
Gesamt	4.59	0.03	1250

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

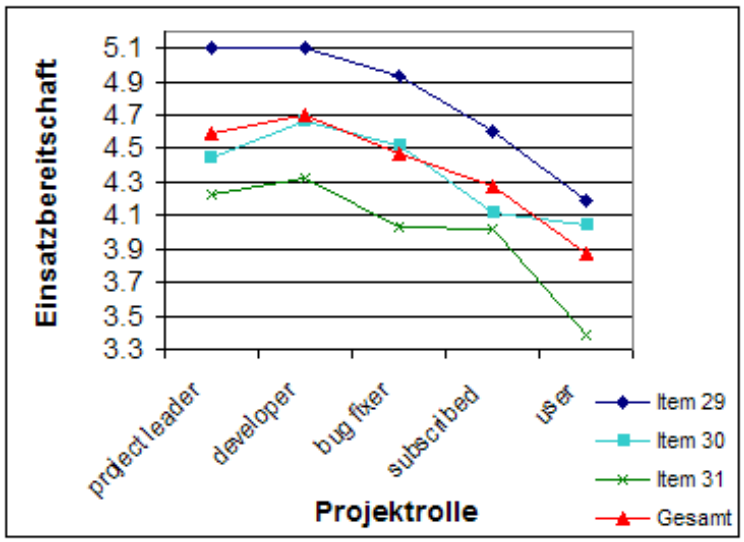


Abbildung 7.7: Einsatzbereitschaft als Funktion der Projekt-Rolle

Einsatzbereitschaft in Abhängigkeit der Anzahl der Hobbys

Die Bereitschaft für zukünftiges Engagement zeigt keine Abhängigkeit von der Anzahl der Hobbys, welche der Open-Source-Entwickler hat. Eine Ausnahme bildet hier das Item 31. Die Bereitschaft, eine zusätzliche Stunde Zeit am Tag für Open Source zu verwenden, nimmt mit einer steigenden Anzahl von Hobbys signifikant ab (α = 5%, Abbildung 7.8). Dieser Befund macht Sinn: Je mehr Hobbys der Open-Source-Entwickler hat, desto mehr valable Alternativen hat er, diese Zeit einzusetzen.

Einsatzbereitschaft und Beschäftigungsgrad

Die Einsatzbereitschaft für Open Source steigt mit sinkendem Beschäftigungsgrad. Dieser Zusammenhang ist ausgeprägt vorhanden für Item 30 (Bereitschaft für grösseres Engagement in der Zukunft) sowie für den Gesamtindex (Abbildung 7.9), wo der Unterschied zwischen einer vollbeschäftigten und einer studierenden Person hochsignifikant ist. Für die Items 29 und 31 ist dieser Unterschied nur schwach signifikant.

Einsatzbereitschaft und Open-Source-Erfahrung

Werden Einsatzbereitschaft und die Erfahrung als Open-Source-Entwickler korreliert, so resultieren negative Koeffizienten. Dieses Resultat ist nachvollziehbar: Je länger ein Programmierer schon aktiv ist, desto grösser ist seine Neigung, das Engagement abzubauen. Die Korrelation ist allerdings nur schwach und einzig für das Item 30 signifikant (Korrelationskoeffizient: -0.08, Signifikanzniveau α: 1%).

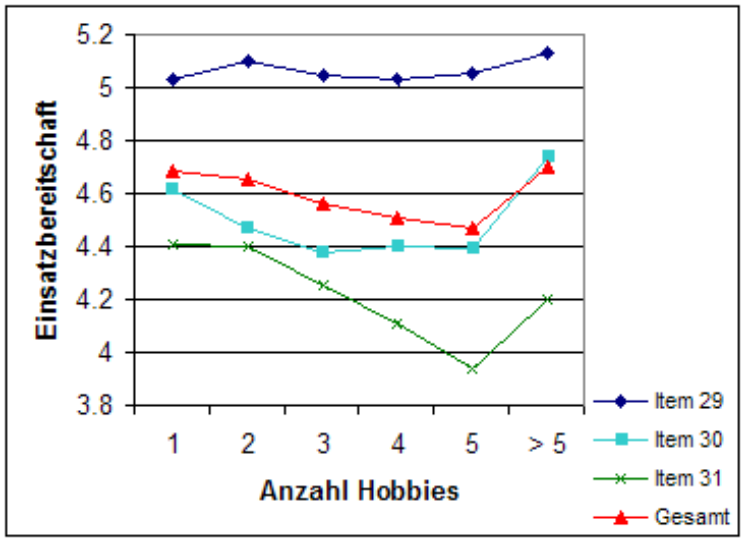


Abbildung 7.8: Einsatzbereitschaft in Abhängigkeit der Anzahl der Hobbys

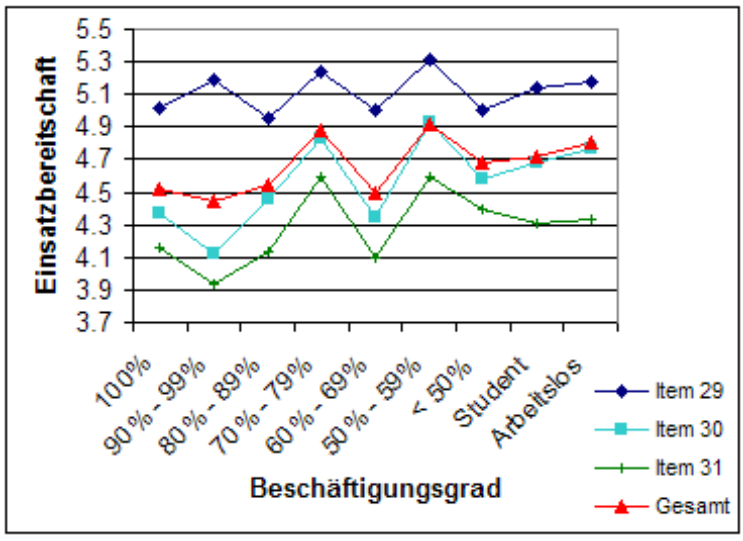


Abbildung 7.9: Einsatzbereitschaft und Beschäftigungsgrad

Tabelle 7.32: Output

	Output pro Jahr		
	Mittelwert	**Std.fehler**	**Anzahl**
Anzahl Patches pro Jahr	28.46	4.55	1187
Anzahl Klassen/Module/Files pro Jahr	73.46	8.68	1193

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

7.2.3 Output und Produktivität

Während zeitliches Engagement die aktuelle und Einsatzbereitschaft die zukünftige Ausprägung von Engagement darstellen, werden mit Output und Produktivität Aspekte von Engagement erhoben, die in der Vergangenheit liegen. Ausgangspunkt für die Berechnung von Output und Produktivität sind die Fragen nach der Anzahl bisher erstellten Patches (Item 38) bzw. Klassen/Modules/Files (Item 39). Ähnlich wie bei Fragen nach der Anzahl produzierter Befehlszeilen ist es auch bei diesen Fragen zweifelhaft, ob mit ihnen ein gültiges Mass für Produktivität konstruiert werden kann. Mit den folgenden Analysen soll untersucht werden, ob solche Zweifel angebracht sind oder im besseren Fall ausgeräumt werden können.

Output

Die Fragen *38* und *39* geteilt durch die Anzahl der Jahre, welche die Person aktiv im Open-Source-Bereich ist, geben ein grobes Mass für den jährlichen Output des Software-Entwicklers (Tabelle 7.32).

Welche Auswirkung hat die Rolle im Open-Source-Projekt auf den jährlichen Output bzw. die Produktivität? Nahe liegend ist die Hypothese, dass der Output umso grösser ist, je wichtiger die Rolle ist (bzw. je war).

Die Analyse der Mittelwerte gruppiert nach Rolle im Open-Source-Projekt unterstützt diese Hypothesen (siehe Tabelle 7.33 sowie die Abbildungen 7.10 und 7.11). Die Unterschiede zwischen den Mittelwerten der Projektleiter, Entwickler und Bug-Fixer sind statistisch signifikant (Signifikanzniveau $\alpha = 1\%$).

Der Output pro Jahr wird grösser, je mehr Zeit der Entwickler für seine Open-Source-Aktivitäten einsetzt. Dieser nachvollziehbare Zusammenhang kann durch eine Korrelationsanalyse eindeutig bestätigt werden (Korrelationskoeffizient: 0.12, Signifikanzniveau α: 1%). Hingegen kann kein Zusammenhang zwischen dem jährlichen Output und der Anzahl der Jahre mit Open-Source-Erfahrung gefunden werden.

Produktivität

Wird der Output pro Jahr mit dem wöchentlichen Zeiteinsatz für Open Source (Frage *40*) in Beziehung gesetzt, so kann eine Abschätzung für die Produktivität der Entwickler berechnet werden (Tabelle 7.34).

Tabelle 7.33: Output als Funktion der Projekt-Rolle

	Output pro Jahr		
	Mittelwert	**Std.fehler**	**Anzahl**
38: Anzahl Patches pro Jahr			
project leader	35.24	6.75	770
developer	20.20	4.44	316
bug fixer	5.13	2.00	41
subscribed to a mailing list	1.17	0.31	35
user	0.32	0.15	19
Gesamt	28.60	4.57	1181
39: Anzahl Klassen/Module/Files pro Jahr			
project leader	100.06	13.09	777
developer	29.44	4.49	319
bug fixer	5.08	2.28	38
subscribed to a mailing list	7.56	5.02	35
user	0.60	0.37	19
Gesamt	73.74	8.71	1188

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

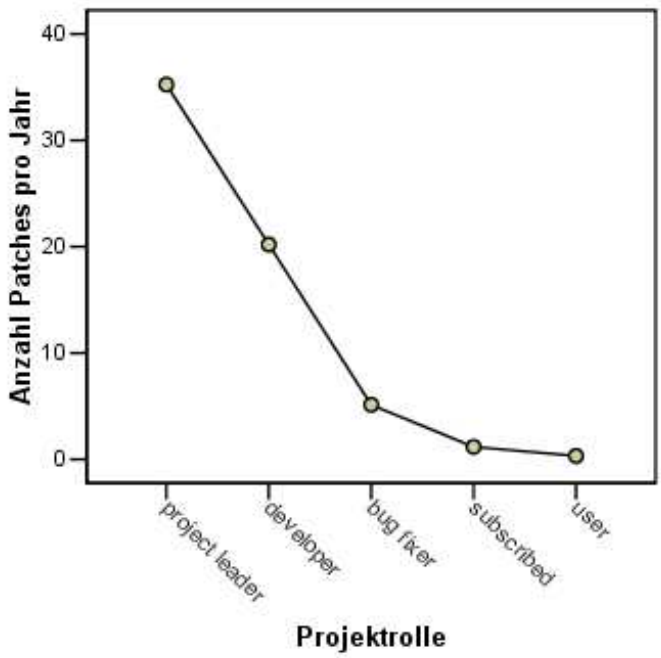


Abbildung 7.10: Output als Funktion der Projekt-Rolle (Patches)

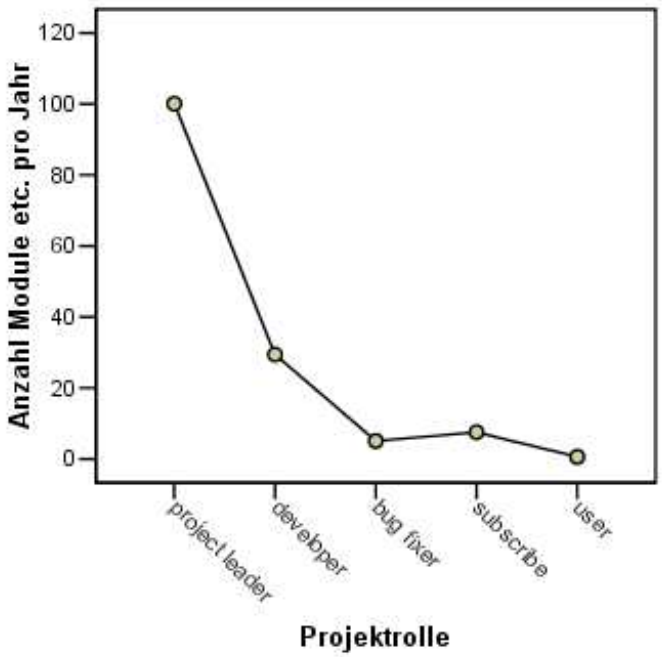


Abbildung 7.11: Output als Funktion der Projekt-Rolle (Module etc.)

Tabelle 7.34: Produktivität

	Output pro Zeiteinheit		
	Mittelwert	**Std.fehler**	**Anzahl**
Produktivität bzgl. Patches	2.55	0.35	1168
Produktivität bzgl. Klassen/Module/Files	7.88	0.99	1172

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.35: Produktivität als Funktion der Projekt-Rolle

	Output pro Zeiteinheit		
	Mittelwert	**Std.fehler**	**Anzahl**
38: Produktivität bzgl. Patches			
project leader	2.85	0.5	760
developer	2.38	0.54	314
bug fixer	1.29	0.45	37
subscribed to a mailing list	0.43	0.2	34
user	0.08	0.06	17
Gesamt	2.56	0.36	1162
39: Produktivität bzgl. Klassen/Module/Files			
project leader	10.5	1.5	762
developer	3.56	0.39	316
bug fixer	2.07	0.94	36
subscribed to a mailing list	0.8	0.27	35
user	0.15	0.09	18
Gesamt	7.91	0.99	1167

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Wieder lässt sich nach den Bedingungen für die Produktivität von Open-Source-Entwicklern fragen. Wird die Produktivität als Funktion der Projekt-Rolle dargestellt, ergibt sich wieder der nahe liegende Zusammenhang von steigender Produktivität, je wichtiger die Rolle ist (Tabelle 7.35 und Abbildung 7.12). Dabei ist allerdings nur der Unterschied in der Produktivität bzgl. Klassen/Module/Files zwischen Projektleitern und Programmierern statistisch signifikant (α = 1%).

Auf der Hand liegend ist auch ein Zusammenhang zwischen Produktivität und der Erfahrung als Open-Source-Entwickler: Mit zunehmender Erfahrung sollte die Produktivität ansteigen. Eine Korrelationsanalyse kann diese Annahme allerdings nicht bestätigen. Es kann kein signifikanter Zusammenhang zwischen diesen beiden Grössen festgestellt werden.

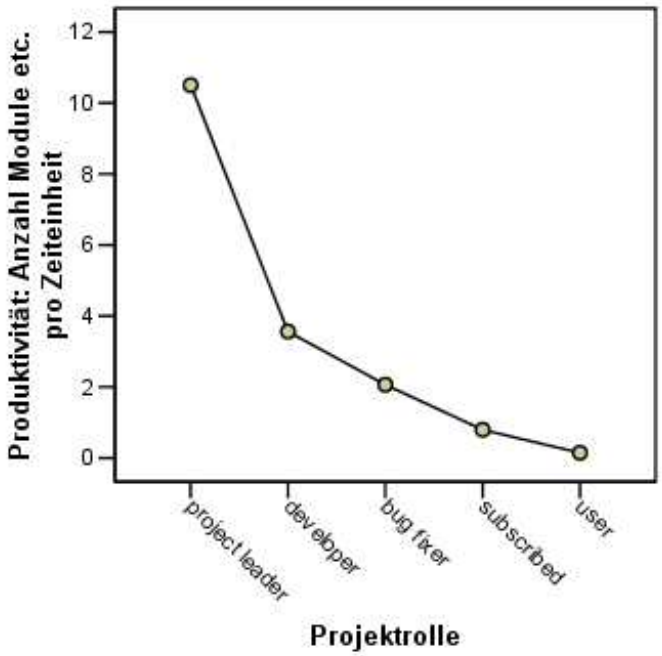


Abbildung 7.12: Produktivität als Funktion der Projekt-Rolle (Klassen etc.)

7.3 Flow-Faktoren

Die abhängige Variable *Spass* in meinem Modell habe ich mit Hilfe des Flow-Konzepts von Csikszentmihalyi operationalisiert. In diesem Kapitel analysiere ich die den Flow-Aspekt betreffenden Daten und untersuche den Zusammenhang von Flow mit anderen Faktoren des Open-Source-Entwicklungsmodells.

7.3.1 Faktorenanalyse der Flow-Items

Voraussetzungen für eine Faktorenanalyse

Die ersten 28 Fragen der Online-Umfrage betreffen das Flow-Erleben der Open-Source-Programmierer. In Anlehnung an den Flow-Fragebogen von Remy (2002) wurden die Probanden nach ihren Empfindungen beim bzw. ihren Einstellungen zum Programmieren befragt. Auf die Frage „Wie oft treffen die folgenden Aussagen auf Sie zu?“ konnten die Open-Source-Entwickler ihre Zustimmung zu den einzelnen Aussagen auf einer 6-stufigen Ratingskala ausdrücken (Werte von *nie* bis *immer*). Mit einem hohen Rating konnten die Probanden eine grosse Häufigkeit zum Ausdruck bringen. Unter den Fragen befanden sich fünf Items, die im Sinne des Flow-Erlebens umgekehrt bewertet waren (z.B. *Frage 15*: „Ich bin mit meinen Gedanken ganz woanders“). Für die Auswertung wurden diese Items umgepolt.

Es stellt sich nun die Frage, ob die abgefragten Variablen auf wenige zugrunde liegende Faktoren reduziert werden können. Im Idealfall sollte eine Faktorenanalyse diese 28 Variablen beispielsweise auf die in Tabelle 4.1 aufgeführten sechs Kategorien abbilden.

Eine einführende Untersuchung der Variablen und ihrer Korrelationen untereinander zeigt, dass die Variablen für eine Faktorenanalyse gut geeignet sind. Der Bartlett-Test auf Sphärizität weist einen hochsignifikanten Wert auf (Signifikanzniveau $\alpha < .1\%$). Auch der KMO-Wert ist mit .897 nahezu hervorragend.

Faktorenanalyse mit sechs Faktoren

Als Methode der Faktorenanalyse wurde die Hauptkomponentenmethode gewählt. Für die Anzahl der extrahierten Faktoren wurde das Kaiser-Kriterium angewendet, gemäss welchem mit den Faktoren weiter gearbeitet wird, welche einen Eigenwert > 1 aufweisen.

Mit einer auf diese Weise konfigurierten Faktorenanalyse konnten sieben Faktoren ermittelt werden, die insgesamt 59.3% der ursprünglichen Varianz erklären. Ein Screeplot zeigt, dass sieben Faktoren das gewählte Kriterium (Eigenwert > 1) erfüllen.

Die Anti-image Korrelations-Matrix weist für alle Frage-Items gute bis sehr gute Werte aus mit Ausnahme der Fragen 13 und 23, welche einen miserablen bzw. zweitklassigen Wert besitzen. Dieser Umstand legte eine revidierte Faktorenanalyse nahe: Die problematischen Frage-Items werden aus der Analyse eliminiert, die Anzahl der einbezogenen Frage-Items reduziert sich somit auf 26. Der KMO-Wert dieser Transformation steigt leicht auf .903, während der Bartlett-Test hochsignifikant bleibt.

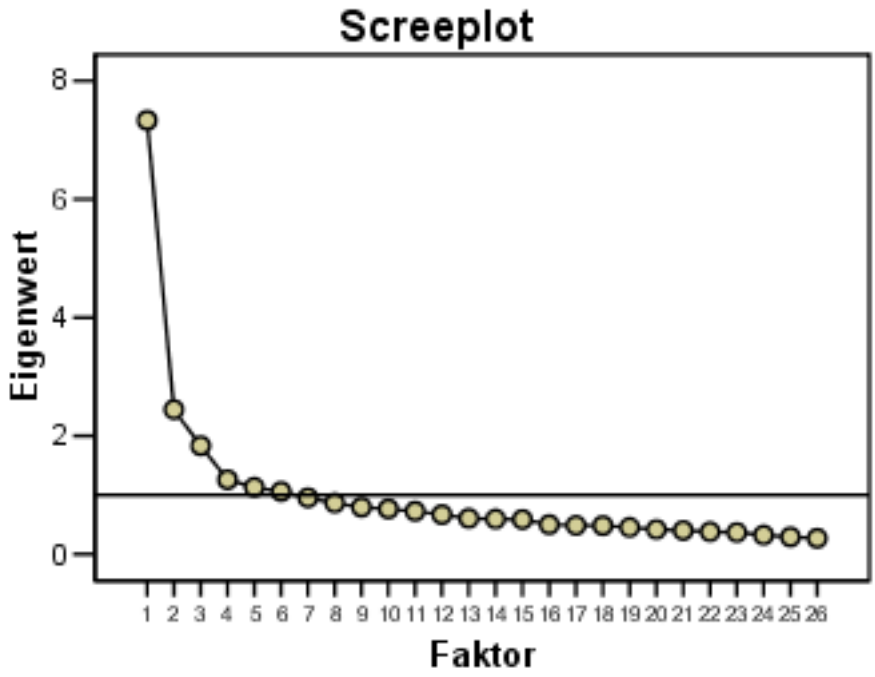


Abbildung 7.13: Screeplot der Faktorenanalyse mit 26 Frage-Items

Diese Faktorenanalyse führt zu einer Lösung mit sechs Faktoren (siehe Screeplot 7.13). Diese Faktoren vermögen insgesamt knapp 58% der ursprünglichen Varianz zu erklären, womit der Erklärungsgehalt um 1.3% unter der Lösung mit sieben Faktoren liegt. Die Anti-image Korrelations-Matrix dieser Lösung weist als schlechtesten KMO-Wert einen Betrag von 0.697 auf, was als akzeptabel gilt.

Die Tabelle mit den Kommunalitäten (siehe 7.36) zeigt, dass die sechs Faktoren minimal 40% (Frage 24) und maximal 70% (Frage 10) der Varianz jedes einzelnen Items erklären.[5]

Tabelle 7.36: Kommunalitäten der rotierten Lösung

1	I lose my sense of time.	68.3%
2	I cannot say how long I've been with programming.	68.0%
3	I am in a state of flow when I'm working.	48.2%
4	I forget all my worries when I'm working.	49.3%
5	It's easy for me to concentrate.	49.1%
6	I'm all wrapped up in the action.	51.9%
7	I am absolutely focused on what I'm programming.	67.7%
8	The requirements of my work are clear to me.	66.6%
9	I hardly think of the past or the future.	52.3%
10	I know exactly what is required of me.	70.5%

[5]Mittelwert und Standardabweichung der Items sind in Tabelle 7.21 aufgelistet.

Tabelle 7.36: Kommunalitäten der rotierten Lösung (Forts.)

11	There are many things I would prefer doing. (-)	50.4%
12	I feel that I can cope well with the demands of the situation.	57.5%
14	I always know exactly what I have to do.	62.6%
15	I'm very absent-minded. (-)	58.3%
16	I don't have to muse over other things.	65.6%
17	I know how to set about it.	54.3%
18	I'm completely focused.	65.7%
19	I feel able to handle the problem.	56.2%
20	I am extremely concentrated.	67.8%
21	I'm looking forward to my programming work.	57.4%
22	I enjoy my work.	56.9%
24	Things just seem to fall into place.	39.7%
25	I forget everything around me.	58.7%
26	I accomplish my work for its own sake.	45.1%
27	I completely concentrate on my programming work.	69.9%
28	I am easily distracted by other things. (-)	48.1%

Um die Ladungen auf den einzelnen Faktoren zu maximieren, wurde die Lösung mit den sechs Faktoren mit der VARIMAX-Methode rotiert. Die rotierte Lösung erleichtert die Interpretation der einzelnen Faktoren. Die Interpretation dieser Lösung ist in Tabelle 7.37 ausgeführt.

Bei der Analyse der ermittelten Faktoren fällt auf, dass der Faktor mit der kleinsten Anzahl Items aus zwei umgekehrt bewerteten Items besteht. Dies lässt die Vermutung zu, dass es sich bei diesem Faktor, den ich mit „umgekehrte Bewertung" bezeichne, um ein *Methodenartefakt* handelt. Dieser Eindruck wird bestärkt durch die Tatsache, dass die zwei eliminierten Items ebenfalls umgekehrt bewerteten Fragen waren. Tatsächlich laden bei einer Faktorenanalyse mit sieben Faktoren diese beiden Items ebenfalls auf den Faktor „umgekehrte Bewertung". Möglicherweise führt die in diesen Fragen enthaltenen Negation zu einem Effekt, der andere, mehr inhaltliche Feinheiten übersteuert.

Die Interpretation der übrigen Faktoren fällt, vor allem mit der Tabelle 4.1 im Hinterkopf, relativ leicht. Der erste Faktor enthält die sieben Items, welche einen Zustand beschreiben, in welchem *Konzentration* wie von selbst kommt und die Kognition ausgeblendet wird. Der zweite Faktor wird durch sechs Items gebildet, welche den Aspekt der Kontrolle und der *Eindeutigkeit* der Handlungsanforderung betreffen. Der nächste Faktor enthält die Items, welche das *Flow-Gefühl* und die Freude an der Tätigkeit ausdrücken. Der vierte Faktor enthält vier Items, welche das *Zeiterleben* beim Programmieren und das Versinken in der Tätigkeit beschreiben.

Der fünfte Faktor wird durch zwei Items gebildet, welche mit der *Aufmerksamkeit* des Software-Entwicklers zu tun haben. Inhaltlich ist dieser Faktor schwierig vom ersten Faktor abzugrenzen. Während in den Items des Faktors „Konzentration" das Subjekt mit statischen Begriffen dargestellt wird (es *ist* bei der Sache bzw. es

ist konzentriert), beschreiben die beiden Items des Faktors „Aufmerksamkeit“ das Subjekt in Aktion beim Denken, allerdings in negativer Abgrenzung (das Subjekt denkt *nicht* an gestern und morgen und *nicht* an andere Sachen).

Bemerkenswert beim Faktor „Flow/Spass“ ist, dass das Item 3 „Die Handlung erfolgt wie aus einem Guss“ mit 42% auf den Faktor „Zeitempfinden/Immersion“ lädt, was nur 2% unter der Ladung auf „Flow/Spass“ liegt. Umgekehrt weist das Item 4 „Ich vergesse meine Sorgen“ eine Faktorladung von 37% auf „Flow/Spass“ auf, was nur 4% unter dessen Ladung auf „Zeitempfinden/Immersion“ liegt. Die Unterschiede in den Faktorladungen der übrigen Items sind deutlich grösser (mindestens 10%).

Tabelle 7.37: Interpretation der Faktoren

	Faktoren	**Ladung**
Konzentration		
Cronbachs α: 0.87, mittlere Inter-Item Korr.: 0.49		
5	It’s easy for me to concentrate.	0.598
6	I’m all wrapped up in the action.	0.571
7	I am absolutely focused on what I’m programming.	0.761
18	I’m completely focused.	0.729
20	I am extremely concentrated.	0.757
27	I completely concentrate on my programming work.	0.772
28	I am easily distracted by other things. (-)	0.611
Eindeutigkeit der Aufgabe		
Cronbachs α: 0.83, mittlere Inter-Item Korr.: 0.45		
8	The requirements of my work are clear to me.	0.762
10	I know exactly what is required of me.	0.802
12	I feel that I can cope well with the demands of the situation.	0.549
14	I always know exactly what I have to do.	0.758
17	I know how to set about it.	0.564
19	I feel able to handle the problem.	0.558
Flow/Spass		
Cronbachs α: 0.65, mittlere Inter-Item Korr.: 0.29		
3	I am in a state of flow when I’m working.	0.443
21	I’m looking forward to my programming work.	0.669
22	I enjoy my work.	0.652
24	Things just seem to fall into place.	0.448
26	I accomplish my work for its own sake.	0.578
Zeitempfinden/Immersion		
Cronbachs α: 0.73, mittlere Inter-Item Korr.: 0.40		
1	I lose my sense of time.	0.808
2	I cannot say how long I’ve been with programming.	0.797
4	I forget all my worries when I’m working.	0.415
25	I forget everything around me.	0.525
Aufmerksamkeit		
Cronbachs α: 0.41, mittlere Inter-Item Korr.: 0.26		

Tabelle 7.37: Interpretation der Faktoren (Forts.)

Faktoren		Ladung
9	I hardly think of the past or the future.	0.594
16	I don't have to muse over other things.	0.777
umgekehrte Bewertung		
Cronbachs α: 0.26, mittlere Inter-Item Korr.: 0.15		
11	There are many things I would prefer doing. (-)	0.613
15	I'm very absent-minded. (-)	0.642

Die Qualität von Skalen, wie sie beispielsweise mit einer Faktorenanalyse gewonnen werden können, kann mit einer Reliabilitätsanalyse abgeschätzt werden. In solchen Reliabilitätsanalysen werden als Masse für die Reliabilität Cronbachs α und die Inter-Item Korrelation berechnet. Im Falle sozialwissenschaftlicher Studien gelten Skalen als ausreichend reliabel, wenn sie Werte für Cronbachs α grösser als 0.7 aufweisen (siehe Bland und Altman (1997)).

In meinem Fall wird dieser Grenzwert durch die Faktoren „Konzentration", „Eindeutigkeit der Aufgabe" und „Zeitempfinden/Immersion" klar überboten, während der Faktor „Flow/Spass" knapp unter diesem Grenzwert liegt. Dagegen weisen die Faktoren „Aufmerksamkeit" und „umgekehrte Bewertung" mit Cronbachs α von 0.41 bzw. 0.26 indiskutable Werte auf. Die miserable Reliabilität des Faktors „umgekehrte Bewertung" bestätigt den Eindruck, dass es sich hier um einen Befragungsartefakt handelt. Der schlechte Wert des Faktors „Aufmerksamkeit" bezüglich der Reliabilität scheint mir das Abgrenzungsproblem bei der Interpretation zu belegen. Auf Grund dieser schlechten Reliabilitätswerte werde ich diese beiden Faktoren in den weiteren Berechnung nicht mehr berücksichtigen.

Damit verbleiben für die weitere Untersuchung die vier Faktoren „Konzentration", „Eindeutigkeit der Aufgabe", „Flow/Spass" und „Zeitempfinden/Immersion". Das Resultat der Faktorenanalyse ergibt somit eine recht gute Übereinstimmung mit den in der Tabelle 4.1 aufgeführten Flow-Komponenten.

7.3.2 Bedingungen für Flow-Erleben

Welche Bedingungen fördern das Erleben von Flow beim Entwickeln von Open-Source-Software? In den folgenden Abschnitten untersuche ich einige Abhängigkeiten des Flow-Erlebens von anderen Faktoren des Open-Source-Entwicklungsmodells.

Zusammenhang Flow und Projekt-Rolle

Gibt es einen Zusammenhang zwischen Flow-Empfinden und der Rolle, die ein Open-Source-Entwickler in den Projekten einnimmt? Eine Analyse der Mittelwerte der rotierten Flow-Faktoren in Abhängigkeit der Projekt-Rolle weist nur in einem Fall signifikante Differenzen auf. Die Abbildung 7.14 zeigt, dass Projektleiter und Entwickler deutlich mehr Spass und Flow erleben als die anderen Beteiligten

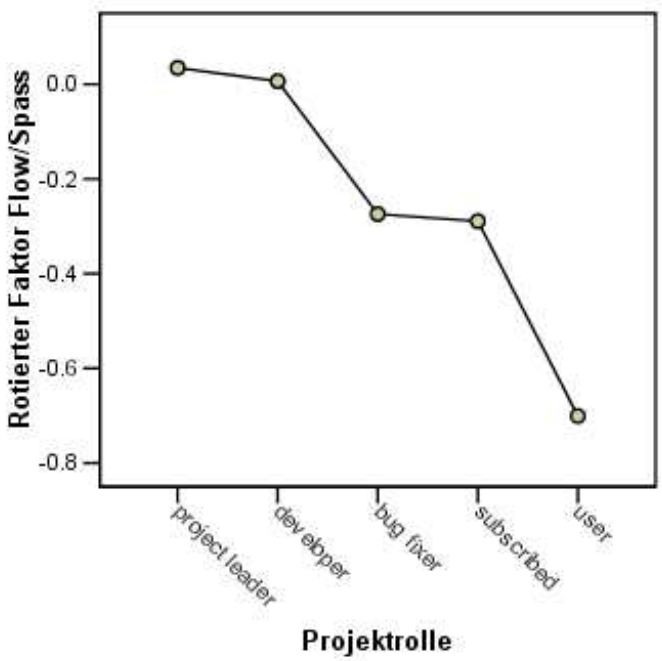


Abbildung 7.14: Flow/Spass als Funktion der Rolle in Open-Source-Projekten

(siehe auch Tabelle 7.38). Dieser Unterschied ist hochsignifikant (Signifikanzniveau α=1%).

Korrelationen zwischen Flow und Engagement

Besteht ein Zusammenhang zwischen Flow-Empfinden und dem zeitlichen Engagement für Open Source? Eine Korrelations-Analyse zwischen zeitlichem Engagement und den rotierten Flow-Faktoren zeigt in mehreren Fällen einen hochsignifikanten Zusammenhang.

Wird der Zeiteinsatz allgemein korreliert, so fällt der Zusammenhang mit den Faktoren „Eindeutigkeit der Aufgabe“, „Flow/Spass“ und „Zeitempfinden/Immersion“ hochsignifikant aus. Wie erwartet ist die Korrelation positiv (Tabelle 7.39).

Interessant ist der Vergleich des zeitlichen Engagements für Open Source in der Freizeit bzw. während der Arbeitszeit (Tabellen 7.40 und 7.41). Der Faktor „Eindeutigkeit der Aufgabe“ korreliert in jedem Fall positiv mit dem zeitlichen Engagement. Wird in der Freizeit programmiert, so korrelieren zusätzlich noch die Faktoren „Konzentration“, „Zeitempfinden/Immersion“ und „Flow/Spass“, wobei die Korrelation in diesen Faktoren hochsignifikant positiv ist. Dies ist ein Hinweis darauf, dass Open-Source-Entwickler mehr Flow erleben, wenn sie diese Tätigkeit in der Freizeit ausüben.

Tabelle 7.38: Flow/Spass als Funktion der Rolle

	Mittelwert	Std.fehler	Anzahl
project leader	0.03	0.04	614
developer	0.01	0.07	253
bug fixer	-0.27	0.17	28
subscribed to a mailing list	-0.29	0.26	32
user	-0.70	0.34	18
Gesamt	-0.01	0.03	945

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.39: Korrelation Flow und Zeiteinsatz

Eindeutigkeit der Aufgabe	0.108^{***}
	(0.001)
Flow/Spass	0.096^{***}
	(0.003)
Zeitempfinden/Immersion	0.095^{***}
	(0.004)

Bemerkung: *** Korrelation auf 1% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.40: Korrelation Flow und Zeiteinsatz in Freizeit

Konzentration	0.086^{***}
	(0.010)
Eindeutigkeit der Aufgabe	0.081^{**}
	(0.015)
Flow/Spass	0.116^{***}
	(0.000)
Zeitempfinden/Immersion	0.133^{***}
	(0.000)

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.41: Korrelation Flow und Zeiteinsatz während Arbeitszeit

Eindeutigkeit der Aufgabe	0.079**
	(0.017)

Bemerkung: ** Korrelation auf 5% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.42: Korrelation Flow und Einsatzbereitschaft

Gesamtindex	
Konzentration	0.152***
	(0.000)
Eindeutigkeit der Aufgabe	0.077**
	(0.019)
Flow/Spass	0.396***
	(0.000)
Zeitempfinden/Immersion	0.219***
	(0.000)

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Korrelationen zwischen Flow und Einsatzbereitschaft

Besteht ein Zusammenhang zwischen Flow-Erleben und der Bereitschaft, für Open Source Zeit und Kreativität zu investieren? Die Korrelation der Flow-Faktoren mit den Antworten zur Einsatzbereitschaft (siehe Kapitel 7.2.2) zeigt einen hochsignifikanten Zusammenhang in den Faktoren „Konzentration“, „Flow/Spass“ und „Zeitempfinden/Immersion“ während die Korrelation mit „Eindeutigkeit der Aufgabe“ noch auf dem 5%-Niveau signifikant ist (siehe Tabelle 7.42). Am meisten ausgeprägt ist die Korrelation des Gesamtfaktors mit der Komponente „Flow/Spass“.

Werden die Fragen zur Einsatzbereitschaft einzeln untersucht, so zeigt sich, dass die Faktoren „Konzentration“, „Flow/Spass“ und „Zeitempfinden/Immersion“ immer hochsignifikant korrelieren, während die Korrelation des Faktors „Eindeutigkeit der Aufgabe“ nur mit der Frage 29 hochsignifikant ausfällt (siehe Tabellen 7.43, 7.44 und 7.45). Der Faktor „Flow/Spass“ korreliert in jedem Fall am stärksten.

Tabelle 7.43: Korrelation Flow und Einsatzbereitschaft (29)

Ich freue mich auf weitere Entwicklungstätigkeiten für Open-Source-Software.	
Konzentration	0.153***
	(0.000)
Eindeutigkeit der Aufgabe	0.085***
	(0.009)
Flow/Spass	0.371***
	(0.000)
Zeitempfinden/Immersion	0.125***
	(0.000)

Bemerkung: *** Korrelation auf 1% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.44: Korrelation Flow und Einsatzbereitschaft (30)

Ich bin bereit, mein zeitliches Engagement bei der Entwicklung von Open-Source-Software in Zukunft zu vergrössern.	
Konzentration	0.085***
	(0.009)
Flow/Spass	0.322***
	(0.000)
Zeitempfinden/Immersion	0.202***
	(0.000)

Bemerkung: *** Korrelation auf 1% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.45: Korrelation Flow und Einsatzbereitschaft (31)

Eine zusätzliche Stunde Zeit im Tag würde ich zum Programmieren von Open Source verwenden.	
Konzentration	0.142***
	(0.000)
Eindeutigkeit der Aufgabe	0.074**
	(0.024)
Flow/Spass	0.295***
	(0.000)
Zeitempfinden/Immersion	0.202***
	(0.000)

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Korrelationen zwischen Flow und Output

Besteht ein Zusammenhang zwischen Flow-Empfinden und dem Output bzw. der Produktivität der Entwickler? Unsere Vermutung ist, dass sich Flow-Empfinden positiv auf diese Grössen auswirkt: Wer mehr Spass hat, ist häufiger und produktiver bei der Arbeit, was sich in einem gesteigerten Output sowohl pro Jahr wie auch pro Zeiteinheit auswirken sollte.

Die Korrelations-Analyse ergibt allerdings nur in einem einzigen Fall eine positive Korrelation: Der Faktor „Eindeutigkeit der Aufgabe“ ist schwach positiv mit dem Output pro Jahr korreliert (Signifikanzniveau $\alpha = 5\%$, siehe Tabelle 7.46). Im Gegensatz zur Studie von Hertel u. a. (2003, siehe S. 27), in welcher die Anzahl Patches positiv und mit signifikanten Werten mit den erhobenen Motivationsfaktoren korrelierten, erhalte ich weniger aussagekräftige Resultate. Es ist zu überlegen, ob die gewählten Masse für die Leistung der Entwickler für eine breite Studie, wie ich sie durchführte, aussagekräftige Ergebnisse liefern können. Hertel u. a. konzentrierten sich nur auf die Entwickler im Linux-Kernel-Projekt. In dieser Gruppe dürfte die Interpretation der Begriffe konsistenter gewesen sein als dies in der von mir untersuchten Population möglich ist.

Weitere Korrelationen mit Flow-Empfinden

Besteht ein Zusammenhang zwischen Spass und Erfahrung? Man könnte sich vorstellen, dass eine Person umso mehr Spass empfindet, je mehr Erfahrung sie im Open-Source-Bereich hat, beispielsweise weil sie dann umso kompetenter mit den Programmier-Werkzeugen umgehen kann.

Um diese Annahme zu testen, wurde eine Korrelations-Analyse durchgeführt mit den Flow-Dimensionen einerseits und der Erfahrung andererseits. Als Varia-

Tabelle 7.46: Korrelation Flow und Output

Output von Klassen/Modulen/Files etc. pro Jahr	
Eindeutigkeit der Aufgabe	0.074**
	(0.027)

Bemerkung: ** Korrelation auf 5% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

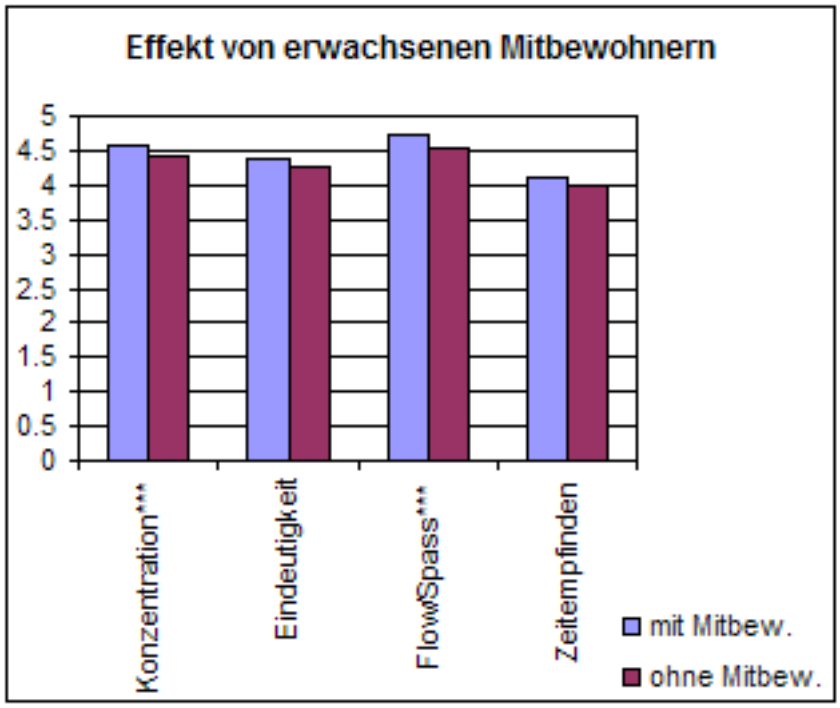
Quelle: Benno Luthiger, FASD Studie, Universität Zürich

blen für die Erfahrung wurde die Anzahl der Jahre im Open-Source-Bereich sowie die Anzahl der Jahre multipliziert mit dem zeitlichen Engagement (Anzahl der Stunden pro Woche für Open Source). Die Analyse ergab nur für eine Kombination einen signifikanten Wert: Die Flow-Komponente „Flow/Spass“ korreliert signifikant (auf 5% Niveau) mit der zusammengesetzten Erfahrung (Jahre mal Wochenstunden).

Nun korreliert aber schon das zeitliche Engagement mit Flow/Spass (siehe Tabelle 7.39), während sich keine Korrelation für die Kombination Flow/Spass und Anzahl der Jahre diagnostizieren lässt. Die oben festgestellte Korrelation kann also mit der Korrelation des Faktors „zeitliches Engagement“, der in die Definition von Erfahrung eingeht, erklärt werden. Auf Grund dieser Überlegungen kann gefolgert werden, dass sich aus den Daten kein Zusammenhang zwischen Erfahrung und Spass-Empfinden ableiten lässt.

Bei einem Vergleich der Mittelwerte der Flow-Komponenten in Abhängigkeit mit der Anzahl der Hobbys weisen die Personen, welche neben dem Programmieren noch zusätzliche Hobbys pflegen, einen signifikant höheren Wert in den Komponenten „Konzentration“ (Signifikanzniveau $\alpha = 1\%$), „Eindeutigkeit der Aufgabe“ und „Flow/Spass“ (je $\alpha = 10\%$) auf gegenüber Entwicklern mit bloss einem Hobby. Dies kann so gedeutet werden, dass eine zu ausschliessliche Fixierung auf das Programmieren dem Spass an der Tätigkeit schadet.

Frauen erleben beim Programmieren von Open-Source-Software nicht mehr oder weniger Spass und Flow als Männer. Nur beim Faktor „Eindeutigkeit der Aufgabe“ weisen sie einen auf dem 5%-Niveau signifikant tieferen Wert auf als männliche Software-Entwickler. Auch Software-Entwickler mit Kindern unterscheiden sich in dieser Hinsicht nicht wesentlich von den kinderlosen Programmierern. Einzig in Bezug auf die Konzentration beim Programmieren scheinen Kinder einen signifikant negativen Effekt zu haben (Signifikanzniveau $\alpha = 5\%$). Eine umgekehrte Wirkung haben im gleichen Haushalt wohnende erwachsene Personen. Programmierer, die in einem Haushalt mit Mitbewohnern leben, weisen in den Komponenten „Konzentration“ und „Flow/Spass“ hochsignifikant höhere Werte auf als ihre allein lebenden Kollegen (siehe Abbildung 7.15).



Bemerkung: *** Korrelation auf 1% Niveau signifikant

Abbildung 7.15: Flow-Empfinden und Mitbewohner

7.4 Typisierung der Open-Source-Beitragsleister

In diesem Kapitel versuche ich, die Open-Source-Entwickler zu typisieren. Die Typisierung nehme ich mit Hilfe einer Clusteranalyse der Fragen zur Motivation der Beitragsleister vor. Im zweiten Teil untersuche ich, wie sich die auf diese Weise gewonnenen Typen bezüglich Spass, Engagement, Einsatzbereitschaft usw. verhalten und differenzieren. Abgesehen vom unmittelbaren Erkenntnisgewinn bezweckt diese vertiefende Untersuchung, die Interpretation der im ersten Schritt gefundenen Typen zu validieren.

7.4.1 Clusteranalyse

Können die Open-Source-Entwickler auf Grund ihrer Motivationen charakterisiert und typisiert werden? Zur Beantwortung dieser Frage wurde mit den Fragen 36 und 37 eine Clusteranalyse durchgeführt. Eine partionierende Clusteranalyse (K-Means-Methode) ergibt das beste Resultat bezüglich Ausgewogenheit der Clustergrösse, Stabilität und Konsistenz mit einer Lösung, welche aus drei Clustern besteht.

Bei der Analyse der Motive von Projektmitarbeitern (Tabelle 7.47) zeigt sich, dass die Motive 5 (*Kollegen haben mich dazu motiviert, am Open-Source-Projekt mitzuarbeiten*) und 2 (*Mein Arbeitgeber beauftragte mich, am Open Source Projekt mitzuarbeiten, weil er die Funktionalität benötigte*) am stärksten zur Unterscheidung der drei Cluster beitragen. Cluster 1 zeichnet sich durch niedrige Werte bei allen ausser dem ersten Motiv aus. Die Entwickler in diesem Cluster können als *extrinsisch-pragmatisch motiviert* bezeichnet werden. Cluster 2 weist überall die höchsten Werte auf, insbesondere bei den Motiven 5 und 2. Die Programmierer in diesem Cluster können als *extrinisch-sozial motiviert* charakterisiert werden. Cluster 3 fällt durch die niedrigen Werte in den ersten beiden Motiven auf. Die Probanden in diesem Cluster können als *intrinsisch-altruistisch motiviert* bezeichnet werden.

Die Clusterbildung bei der Analyse der Motive der Projektleiter (Tabelle 7.48) wird am stärksten durch die Fragen 4 (*Weil ich etwas für die Open-Source-Community machen wollte*) und 7 (*Ich benötigte Helfer, um ein Software-Projekt fertig stellen zu können*) bestimmt. Cluster 1 zeichnet sich durch die niedrigen Werte in den ersten beiden Fragen aus. Die Projektleiter in diesem Cluster können als *intrinsisch-altruistisch motiviert* beschrieben werden. Cluster 2 ist durch viele vergleichsweise geringe Werte bestimmt, insbesondere bei den Motiven 4 und 7. Die Programmierer in diesem Cluster können als *pragmatisch-individualistisch motiviert* charakterisiert werden. Cluster 3 zeichnet sich durch vergleichsweise hohe Werte bei allen Motiven aus. Die Entwickler in diesem Cluster können als *netzwerk-orientiert motiviert* bezeichnet werden.

Eine Clusteranalyse lässt bei der Interpretation viel Spielraum zu. In einem zweiten Schritt versuchte ich deshalb, mit Hilfe einer Faktorenanalyse den Befund der Clusteranalyse zu erhärten. Dabei ging ich von einer Lösung mit drei Faktoren[6] aus, in der Annahme, dass die so gebildeten Faktoren mit den im ersten Schritt

[6] Screeplot und Kaiser-Kriterium würden in diesem Fall eher für eine Lösung mit zwei Faktoren sprechen.

Tabelle 7.47: Motive als Programmierer in einem Open-Source-Projekt

Motive	**Cluster 1** extr.-pragm.	**Cluster 2** extr.-sozial	**Cluster 3** intrin.-altruistisch
1. It was important for my work to have a certain functionality; that's why I joined the open source project and got involved.	5	5	3
2. My employer asked me to participate in the open source project because he needed its functionality.	2	3	1
3. Because I wanted to do something for the open source community.	3	5	5
4. The project promised to be fun.	3	5	5
5. My colleagues motivated me to participate in the open source project.	2	4	2
6. By participating in the open source project you could become famous.	2	3	3
7. Because I wanted to learn and develop new skills.	4	5	5
Anteil des Clusters im Sample	25%	31%	44%

Bemerkung: n = 1230

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.48: Motive als Projektleiter an einem Open-Source-Projekt

Motive	**Cluster 1** intrin.-altruistisch	**Cluster 2** pragm.-indiv.	**Cluster 3** netzwerk-orientiert
1. I needed a certain functionality for my work, and I wanted to make this functionality available as an open source application.	4	5	5
2. My employer asked me to start the open source project because he needed the functionality.	1	2	3
3. My employer asked me to start the open source project because he could earn money with the application.	1	1	2
4. Because I wanted to do something for the open source community.	5	3	5
5. One open source project in the past didn't develop as desired, therefore I had to start my own.	3	2	5
6. The project promised to be fun.	5	4	5
7. I needed assistants to complete a software project.	3	2	4
8. With an open source project you could become famous.	3	2	3
Anteil des Clusters im Sample	42%	23%	35%

Bemerkung: n = 1068

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.49: Faktorenanalyse der Motive als Programmierer

Faktoren	Ladung
Open-Source-Bewegung	
3 Because I wanted to do something for the open source community.	0.78
4 The project promised to be fun.	0.69
7 Because I wanted to learn and develop new skills.	0.67
soziales Umfeld	
2 My employer asked me to participate in the open source project because he needed its functionality.	0.59
5 My colleagues motivated me to participate in the open source project.	0.79
6 By participating in the open source project you could become famous.	0.60
pragmatischer Nutzen	
1 It was important for my work to have a certain functionality; that's why I joined the open source project and got involved.	0.90

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

gefundenen Clustern übereinstimmen. Bei dieser Analyse interessierte mich nicht in erster Linie die Anzahl der den Motiven zugrunde liegenden Faktoren, sondern wie stark die einzelnen Motive auf welchen Cluster laden.

Die Voruntersuchung, ob sich die Items der Fragen 36 und 37 für eine Faktorenanalyse eignen, ergab zwiespältige Resultate. Gemäss dem Bartlett-Sphärentest waren die Items grundsätzlich gut geeignet für eine solche Analyse (Signifikantniveau = 0.0), die einzelnen Items schnitten gemäss KMO-Mass allerdings jämmerlich bis maximal mittelmässig ab. Da ich aber primär an der Faktorladung interessiert war, taxierte ich diesen Sachverhalt nicht als problematisch.

Die Tabelle 7.49 zeigt die rotierte Komponentenmatrix mit den Faktorladungen der Motive von Programmierern.[7] Die Werte in der Komponentenmatrix legen folgende Interpretation der Faktoren nahe: Auf den ersten Faktor laden die Motive, die stark mit dem Gepräge von Open Source verbunden sind: etwas für die Community tun, Spass haben und neue Fähigkeiten erwerben. Auf den zweiten Faktor laden die Motive, die mit dem sozialen Umfeld des Programmierers zu tun haben: Die motivierenden Kollegen, die Möglichkeit des Erwerbs von Reputation sowie die Veranlassung durch den Arbeitgeber. Der dritte Faktor wird durch das pragmatische Motiv gebildet. Interessant ist, dass das zweite Motivationsitem mit 53% auf den dritten Faktor (den pragmatischen Nutzen) lädt, was nur geringfügig unter seiner Faktorladung von 59% auf dem zweiten Faktor ist. Dies kann so interpretiert werden, dass für pragmatisch motivierte Software-Entwickler auch die pragmatischen Motive des Arbeitgebers relevant sind.

[7] Mit den drei Faktoren können rund 60% der ursprünglichen Variation erklärt werden.

Tabelle 7.50: Clusterbildung mit den Motivationsfaktoren der Programmierer

Motivations-faktor	**Cluster** 1	2	3
Open-Source-Bewegung	0.557	-1.318	0.155
soziales Umfeld	-0.565	-0.341	1.128
pragmatischer Nutzen	-0.239	-0.042	0.402

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.51: Kreuztabelle: Typen von Programmierern

Motivations-faktor	**Typen von Programmierern** extrinsisch-pragmatisch	extrinsisch-sozial	intrinsisch-altruistisch	Gesamt
Open-Source-Bewegung	57	26	443	526
	(135)	(155)	(235)	46.5%
pragmatischer Nutzen	228	1	34	263
	(68)	(78)	(118)	23.3%
soziales Umfeld	6	307	29	342
	(88)	(101)	(153)	30.2%

Bemerkung: erwartete Anzahl in Klammern

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Nachdem ich auf diese Weise die drei Motivationsfaktoren für Programmierer erzeugt hatte, konnte ich mit einer Clusteranalyse die Probanden diesen neu gebildeten Faktoren zuordnen. Die Zuordnung der Cluster zu den Motivationsfaktoren bereitet keinerlei Probleme (siehe Tabelle 7.50). Der erste Cluster weist den höchsten Wert auf dem Faktor „Open-Source-Bewegung“, der zweite auf dem Faktor „pragmatischer Nutzen“, während der dritte Cluster offensichtlich die durch das soziale Umfeld motivierten Programmierer beinhaltet.

Nun galt es noch zu überprüfen, wie gut die mit der Clusteranalyse (siehe Tabelle 7.47) gebildete Typisierung mit den auf die zweite Art gefundenen Typen übereinstimmt. Zu diesem Zweck bildete ich eine Kreuztabelle und verglich die erwartete mit der tatsächlich vorgefundenen Anzahl. Die Kreuztabelle 7.51 zeigt das erwartete Verhalten. Dementsprechend kann die ursprüngliche Interpretation der Typen aufgrund der einfachen Clusteranalyse als bestätigt betrachtet werden. Folgender zusätzlicher Erkenntnisgewinn wurde mit der Faktorenanalyse möglich: 1.) Die ursprüngliche Interpretation eines intrinsischen bzw. Spassmotivierten Typs sollte zugunsten eines umfassend von Open Source faszinierten Typs revidiert werden. 2.) Der sozial motivierte Type ist auch stark von Reputationsüberlegungen angereizt. 3.) Beim pragmatisch motivierten Typ spielen die pragmatischen Überlegungen des Arbeitgebers ebenfalls eine Rolle.

Tabelle 7.52: Faktorenanalyse der Motive als Projektleiter

Faktoren		Ladung
professionelles Umfeld		
2	My employer asked me to start the open source project because he needed the functionality.	0.88
3	My employer asked me to start the open source project because he could earn money with the application.	0.85
Open-Source-Bewegung		
4	Because I wanted to do something for the open source community.	0.55
5	One open source project in the past didn't develop as desired, therefore I had to start my own.	0.58
6	The project promised to be fun.	0.58
7	I needed assistants to complete a software project.	0.62
8	With an open source project you could become famous.	0.55
pragmatischer Nutzen		
1	I needed a certain functionality for my work, and I wanted to make this functionality available as an open source application.	0.85

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Dieses ganze Prozedere mit der Faktorenanalyse führte ich auf die gleiche Weise mit den Motiven für die Projektleiter durch. Die rotierte Komponentenmatrix der Motivationsfaktoren der Projektleiter (Tabelle 7.52) kann ganz ähnlich wie im ersten Fall interpretiert werden.[8] Auf den ersten Faktor laden die Motive, die starkes Gewicht auf die Veranlassung durch den Arbeitgeber legen. Auf den zweiten Faktor laden die Motive, die stark mit dem Gepräge von Open Source verbunden sind: etwas für die Community tun, Spass haben, ein Projekt wieder auf Kurs bringen und mit einem Open-Source-Projekt berühmt werden. Der dritte Faktor wird bestimmt durch das pragmatische Motiv, eine benötigte Funktionalität mit einem eigenen Open-Source-Projekt zu realisieren. In diesem Fall ist es das vierte Motiv (etwas für die Community tun) welches neben der Ladung von 55% auf den Faktor „Open-Source-Bewegung" mit 49% auf „pragmatischer Nutzen" stark auf einen zweiten Faktor lädt. Entweder birgt der Einsatz für die Open-Source-Community einen pragmatischen Aspekt oder dann sind die pragmatisch motivierten Projektleiter doch nicht so pragmatisch, wie es auf den ersten Blick scheinen mag.

Die Interpretation der Cluster, welche mit den Motivationsfaktoren der Projektleiter gebildet wurden, bereitet mehr Mühe als im ersten Fall (Tabelle 7.53). Beim ersten Cluster handelt es sich eindeutig um den professionellen Typ. Die Zuordnung des zweiten Clusters zu einem Motivationsfaktor ist allerdings nicht eindeutig. Den grössten Wert weist dieser Cluster ebenfalls für den Faktor „professionelles Umfeld" auf. Aus diesem Grund griff ich auf den zweithöchsten Wert zurück

[8] Die drei Faktoren erklären rund 57% der ursprünglichen Varianz.

Tabelle 7.53: Clusterbildung mit den Motivationsfaktoren der Projektleiter

Motivations-faktor	**Cluster** 1	2	3
professionelles Umfeld	1.687	-0.131	-0.529
Open-Source-Bewegung	0.373	-1.097	0.461
pragmatischer Nutzen	0.147	-0.329	0.125

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.54: Kreuztabelle: Typen von Projektleitern

Motivations-faktor	**Typen von Projektleitern** intrinsisch-altruistisch	pragmatisch	netzwerk-orientiert	Gesamt
Open-Source-Bewegung	343	2	166	511
	(218)	(120)	(173)	52.7%
pragmatischer Nutzen	70	198	9	277
	(118)	(65)	(94)	28.6%
professionelles Umfeld	1	27	154	182
	(78)	(43)	(62)	18.8%

Bemerkung: erwartete Anzahl in Klammern

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

und wies diesem Cluster das Motiv „pragmatischer Nutzen“ zu. Der dritte Cluster kann hingegen wieder eindeutig identifiziert werden als diejenigen Projektleiter, welche durch den spezifischen Charakter des Open-Source-Entwicklungsmodells motiviert sind.

Die Kreuztabelle 7.54 bestätigt diese Interpretation: In den Zellen, in denen sich die zueinander passenden Typen treffen, übersteigt die tatsächliche Anzahl die erwartete signifikant. Dementsprechend kann die ursprüngliche Interpretation der Typen aufgrund der einfachen Clusteranalyse als bestätigt betrachtet werden. Folgender zusätzlicher Erkenntnisgewinn konnte mit der Faktorenanalyse gewonnen werden: 1.) Die ursprüngliche Interpretation eines intrinsischen bzw. Spass-motivierten Typs sollte zugunsten eines umfassend von Open Source faszinierten Typs revidiert werden. 2.) Der durch das soziale Umfeld zum Open-Source-Engagement motivierte Typ ist in erster Linie ein *professional*, das heisst ein Software-Entwickler, der Open Source im Auftrag des Arbeitgebers entwickelt. 3.) Der pragmatische Typ ist in einem gewissen Mass auch um die Open-Source-Community bemüht.

Wie gross ist nun die Übereinstimmung zwischen den Typen von Programmierern und denjenigen von Projektleitern? Eigentlich sollte ein Proband, welcher z.B. als Programmierer pragmatisch motiviert ist bzw. als solcher typisiert wor-

Tabelle 7.55: Kreuztabelle: Übereinstimmung der Typen

Typen von Programmierern	**Typen von Projektleitern** Open-Source-Bewegung	pragmatischer Nutzen	professionelles Umfeld	Gesamt
Open-Source-Bewegung	301 (220)	90 (118)	28 (81)	419
pragmatischer Nutzen	47 (108)	131 (58)	28 (40)	206
soziales Umfeld	128 (148)	35 (80)	119 (54)	282
Gesamt	476	256	175	907

Bemerkung: erwartete Anzahl in Klammern

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

den ist, auch als Projektleiter diesem Typ angehören. Die Kreuztabelle 7.55 zeigt tatsächlich eine recht grosse Übereinstimmung. Allerdings können auch etliche Abweichungen erkannt werden. Beispielsweise gehören von den 419 Programmierern, welche durch die Open-Source-Bewegung motiviert wurden, 301 Personen auch als Projektleiter zu diesem Typ. Aber schon 90 dieser Programmierer sind als Projektleiter pragmatisch und 28 durch ihr professionelles Umfeld motiviert.

Abschliessend stellt sich die Frage, wie gut diese Typisierung, welche mit Hilfe der Analyse der ursprünglichen Teilnahmemotivation gewonnen worden ist, mit der Aufteilung der Open-Source-Programmierer in Professionals und Open-Source-Hacker übereinstimmt, welche auf Grund des Zeiteinsatzes in der Freizeit vorgenommen wurde (siehe Tabelle 7.20). Nahe liegend ist die Vermutung, dass sich der intrinsisch motivierte, von Open Source faszinierte Typ überdurchschnittlich häufig unter den Open-Source-Hackern findet, während es sich beim durch das berufliche Umfeld motivierten Typ konsequenterweise überdurchschnittlich häufig um einen Professional handelt. Diese Annahmen werden durch die Kreuztabelle 7.56 weitgehend bestätigt. Unter den Professionals befinden sich doppelt so häufig Typen, die durch das professionelle Umfeld motiviert worden sind, während die tatsächliche Anzahl der durch Open Source motivierten Programmierer unter den Hackern die erwartete Häufigkeit ebenso deutlich übertrifft. Die Abweichungen sind auf dem 1%-Niveau signifikant.

7.4.2 Flow-Empfinden der unterschiedlichen Typen

Unterscheiden sich die auf Grund der Clusteranalyse identifizierten Gruppen von Beitragsleistenden in ihrem Empfinden von Flow? Werden für jede Cluster-Gruppe die Mittelwerte der Flow-Faktoren gebildet, so zeigt ein Vergleich, dass tatsächlich für alle Flow-Faktoren signifikante Unterschiede bestehen (Signifikanzniveau $\alpha = 1\%$, siehe Tabelle 7.57).

Tabelle 7.56: Kreuztabelle: Vergleich mit Open-Source-Typen

Open-Source-Typen	**Typen von Projektleitern** Open-Source-Bewegung	pragmatischer Nutzen	professionelles Umfeld	Gesamt
Professional	28	37	36	101
	(53)	(29)	(19)	
Hacker	241	122	27	390
	(205)	(111)	(73)	
Übrige	242	118	119	479
	(252)	(137)	(90)	
Gesamt	511	277	182	970

Bemerkung: erwartete Anzahl in Klammern

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Die durch das Open-Source-Entwicklungsmodell motivierten Programmierer unterscheiden sich von den durch das soziale Umfeld angereizten Software-Entwicklern bezüglich der Flow-Komponenten in keiner Weise. Die Unterschiede in den Mittelwerten befinden sich allesamt im statistischen Streubereich. Hingegen weisen die pragmatisch motivierten Programmierer in allen Flow-Komponenten den geringsten Wert auf (siehe Abbildung 7.16). In den Faktoren „Konzentration“, „Flow/Spass“ und „Zeitempfinden/Immersion“ ist dieser Unterschied hochsignifikant (α = 1%), im Faktor „Eindeutigkeit der Aufgabe“ noch auf den 5%-Niveau signifikant. Dieses Resultat kann auf zweierlei Arten interpretiert werden: Entweder erleben pragmatisch motivierte Software-Entwickler tatsächlich weniger Spass beim Programmieren, oder sie sind generell pragmatisch veranlagt und melden aus diesem Grund tiefere Werte, weil sie ihre Gefühle beim Programmieren nüchterner einschätzen.

Im Falle der Projektleiter weisen die durch das Open-Source-Entwicklungsmodell motivierten Personen in jeder Flow-Komponente den höchsten Wert auf, wie aus der Abbildung 7.17 deutlich abgelesen werden kann (siehe auch Tabelle 7.59). Aus der Tabelle 7.58 wird ersichtlich, dass fast alle Unterschiede im Mittelwert dieses Typs zu den anderen Typen signifikant sind. Hingegen sind die Unterschiede zwischen den pragmatischen und den durch das professionelle Umfeld motivierten Personen bei den Projektleitern nicht mehr signifikant. In diesem Punkt unterscheidet sich das Bild klar von der Situation bei den Programmierern, wo sich die pragmatischen Typen signifikant unterschieden haben. Dies könnte darauf hinweisen, dass bei den aus dem professionellen Umfeld kommenden Open-Source-Projektleitern pragmatische Aspekte eine Rolle spielen.

Die Ergebnisse dieser Analyse zeigen, dass Software-Entwickler, welche von Open Source als Phänomen und Bewegung angesprochen und motiviert werden, signifikant höhere Werte bezüglich Flow und Spass zeigen als pragmatisch motivierte Beitragsleister. Dieses Resultat ist konsistent mit den Erwartungen und kann

Tabelle 7.57: Flow-Empfinden der Typen von Programmierern

Konzentration			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.63	0.03	495
pragmatischer Nutzen	4.36	0.06	251
soziales Umfeld	4.57	0.04	327
Gesamt	4.55	0.02	1073
Eindeutigkeit der Aufgabe			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.36	0.04	461
pragmatischer Nutzen	4.25	0.06	232
soziales Umfeld	4.39	0.04	308
Gesamt	4.34	0.02	1001
Flow/Spass			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.76	0.03	483
pragmatischer Nutzen	4.42	0.05	237
soziales Umfeld	4.74	0.03	321
Gesamt	4.68	0.02	1041
Zeitempfinden/Immersion			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.19	0.04	493
pragmatischer Nutzen	3.80	0.07	246
soziales Umfeld	4.18	0.05	334
Gesamt	4.1	0.03	1073

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

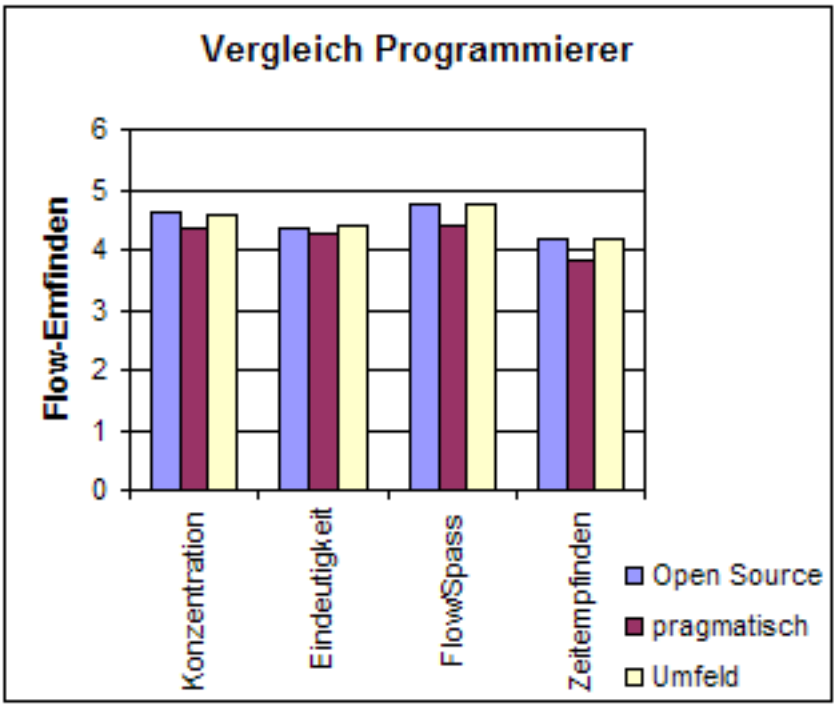


Abbildung 7.16: Flow-Empfinden von Programmierern

Tabelle 7.58: Flow-Komponenten bei Projektleitern: Signifikanz der Unterschiede

	Unterschiede	
Flow-Komponente	Open-Source-Bewegung *zu* pragmatischer Nutzen	Open-Source-Bewegung *zu* professionelles Umfeld
Konzentration	0.00***	0.03**
Eindeutigkeit der Aufgabe	0.06*	-
Flow/Spass	0.00***	0.00***
Zeitempfinden/Immersion	0.00***	0.03**

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.59: Flow-Empfinden der Typen von Projektleitern

Konzentration			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.69	0.03	483
pragmatischer Nutzen	4.43	0.05	269
professionelles Umfeld	4.54	0.06	172
Gesamt	4.59	0.03	924
Eindeutigkeit der Aufgabe			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.42	0.03	457
pragmatischer Nutzen	4.30	0.05	244
professionelles Umfeld	4.36	0.06	165
Gesamt	4.38	0.03	866
Flow/Spass			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.78	0.03	469
pragmatischer Nutzen	4.60	0.05	252
professionelles Umfeld	4.59	0.05	172
Gesamt	4.69	0.02	893
Zeitempfinden/Immersion			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.21	0.04	489
pragmatischer Nutzen	3.98	0.06	262
professionelles Umfeld	4.02	0.07	180
Gesamt	4.11	0.03	931

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

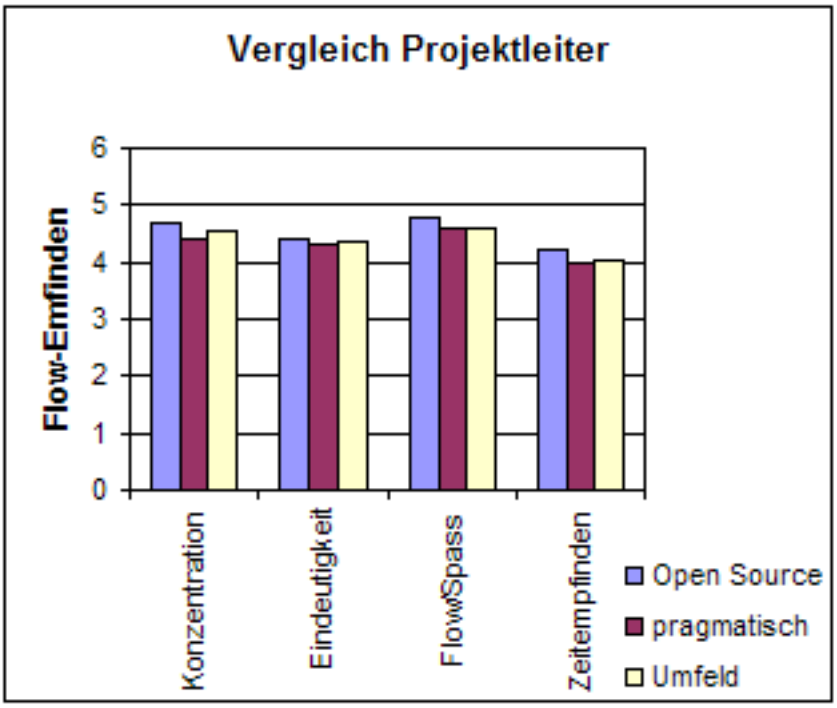


Abbildung 7.17: Flow-Empfinden von Projektleitern

somit als weiterer Hinweis für die Validität der vorgenommenen Typisierung interpretiert werden.

7.4.3 Engagement der unterschiedlichen Typen

Auch bezüglich des zeitlichen Engagements unterscheiden sich die verschiedenen Typen von Beitragsleistern auf signifikante Weise (siehe Tabelle 7.60 für die Werte und Tabelle 7.61 mit den Signifikanzniveaus). Interessant ist der Vergleich des Zeiteinsatzes während der Arbeitszeit mit dem Engagement in der Freizeit. Dieser Vergleich zeigt, dass an Open Source als solches interessierte Programmierer ihren Zeiteinsatz deutlich differenzieren und fast 5 Stunde pro Woche mehr in der Freizeit entwickeln als in der Arbeitszeit (Signifikanzniveau $\alpha = 1\%$). Bei pragmatisch sowie vom Umfeld motivierten Programmierern ist kein solcher Unterschied sichtbar (siehe Abbildung 7.18).

Ein interessantes Bild ergibt sich bei der Analyse des zeitlichen Engagements der verschiedenen Typen von Projektleitern (Tabelle 7.62). Auch bei den Projektleitern engagieren sich die von Open Source faszinierten Typen vornehmlich in der Freizeit (Differenz auf 1%-Niveau signifikant), während die durch das professionelle Umfeld motivierten Projektleiter das grösste Engagement während der Arbeitszeit zeigen. Abbildung 7.19 zeigt weiter, dass der pragmatisch motivierte Typ sich im Freizeiteinsatz gleich wie der professionelle Projektleiter verhält, während er bezüglich des Zeiteinsatzes während der Arbeitszeit das gleiche Engagement

Tabelle 7.60: Zeiteinsatz der Typen von Programmierern

Zeiteinsatz insgesamt			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	11.74	0.52	511
pragmatischer Nutzen	11.56	1.00	243
soziales Umfeld	15.05	0.90	328
Gesamt	12.71	0.43	1082
Zeiteinsatz in Freizeit			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	8.26	0.38	508
pragmatischer Nutzen	5.85	0.42	240
soziales Umfeld	7.26	0.46	326
Gesamt	7.42	0.25	1074
Zeiteinsatz während Arbeitszeit			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	3.48	0.32	508
pragmatischer Nutzen	5.79	0.90	240
soziales Umfeld	7.67	0.72	326
Gesamt	5.27	0.34	1074

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

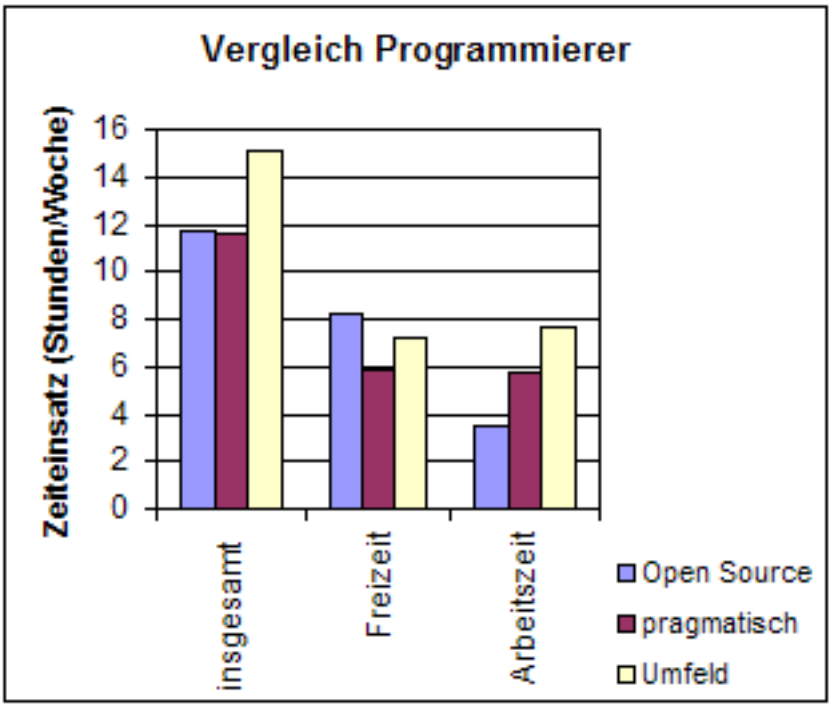


Abbildung 7.18: Zeiteinsatz von Programmierern

Tabelle 7.61: Zeiteinsatz von Programmierern: Signifikanz der Unterschiede

	Unterschiede		
Zeiteinsatz	Open Source *zu* pragmatisch	Open Source *zu* Umfeld	pragmatisch *zu* Umfeld
insgesamt	-	0.00^{***}	0.01^{***}
in Freizeit	0.00^{***}	0.09^{*}	0.02^{**}
während Arbeitszeit	0.02^{**}	0.00^{***}	-

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant
Die Werte in Klammern geben das Signifikanzniveau an (zweiseitig)

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

wie der Open-Source-Typ an den Tag legt. Die Differenzen zwischen Freizeiteinsatz und Zeiteinsatz während der Arbeitszeit sind für die pragmatisch motivierten sowie durch das Umfeld motivierten Projektleiter auf dem 5%-Niveau signifikant.

Diese Analyse legt die Interpretation nahe, dass es sich beim ersten Typ um den klassischen Open-Source-Hacker handelt, welcher von Open Source fasziniert ist und diese Begeisterung in der Freizeit auslebt. Der durch das Umfeld motivierte Typ arbeitet als Projektleiter professionell, d.h. in erster Linie während der Arbeitszeit. Der pragmatisch motivierte Typ zeigt, vor allem als Projektleiter, das kleinste Engagement. Ihm geht es darum, eine Lösung zu realisieren, und das Engagement wird so lange aufrechterhalten, bis dieses Ziel erreicht ist.

7.4.4 Einsatzbereitschaft der unterschiedlichen Typen

Bezüglich der Einsatzbereitschaft zeigen sowohl die Typen von Programmierern wie auch von Projektleitern praktisch das gleiche Verhalten (Tabelle 7.64). Während sich die Einsatzbereitschaft von Personen, die durch Open Source motiviert sind, nicht von derjenigen der durch das Umfeld motivierten Beitragsleistern unterscheidet, gibt es einen hochsignifikanten Unterschied (α = 1%) zu den pragmatisch motivierten Personen (siehe auch Abbildungen 7.20 und 7.21). Wie beim Zeiteinsatz macht sich auch hier wieder die nüchterne Haltung des pragmatisch motivierten Typs deutlich als geringere Einsatzbereitschaft bemerkbar.

7.4.5 Einschätzung der Projektarbeit

Die verschiedenen Typen unterscheiden sich auch bezüglich ihrer Einschätzung der Projektarbeit in mancherlei Weise (Tabellen 7.65 und 7.66). Ins Auge sticht, dass der durch das Umfeld motivierte Programmierer hochsignifikant häufiger Abgabetermine verspürt (Item 33) als die anderen Typen (α = 1%, siehe Abbildungen 7.22 und 7.23). Dieses Merkmal ist im Falle von Projektleitern sogar noch ausgeprägter. Dieses Resultat stützt eindeutig die Interpretation, dass es sich beim Umfeld der Projektleiter um das berufliche Umfeld handelt und dass sich diese Beitragsleister

Tabelle 7.62: Zeiteinsatz der Typen von Projektleitern

Zeiteinsatz insgesamt			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	13.65	0.60	501
pragmatischer Nutzen	10.68	0.89	261
professionelles Umfeld	15.51	1.31	175
Gesamt	13.17	0.47	937
Zeiteinsatz in Freizeit			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	9.21	0.42	497
pragmatischer Nutzen	6.25	0.44	260
professionelles Umfeld	6.12	0.53	175
Gesamt	7.80	0.28	932
Zeiteinsatz während Arbeitszeit			
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.42	0.36	497
pragmatischer Nutzen	4.44	0.75	260
professionelles Umfeld	9.39	1.18	175
Gesamt	5.36	0.36	932

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

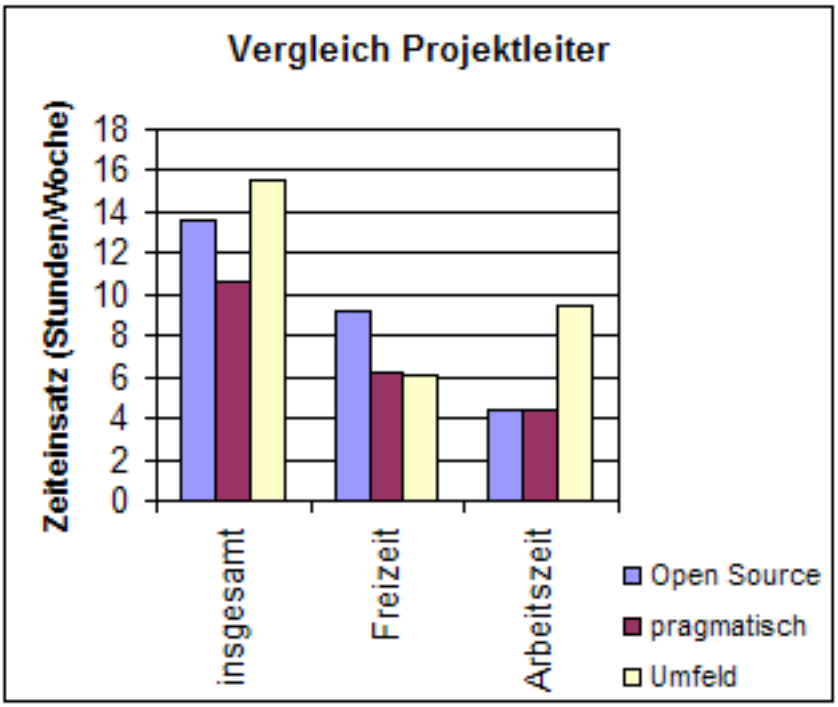


Abbildung 7.19: Zeiteinsatz von Projektleitern

Tabelle 7.63: Zeiteinsatz von Projektleitern: Signifikanz der Unterschiede

	Unterschiede		
Zeiteinsatz	Open Source *zu* pragmatisch	Open Source *zu* Umfeld	pragmatisch *zu* Umfeld
insgesamt	0.00^{***}	-	0.01^{***}
in Freizeit	0.00^{***}	0.00^{***}	-
während Arbeitszeit	-	0.00^{***}	0.00^{***}

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

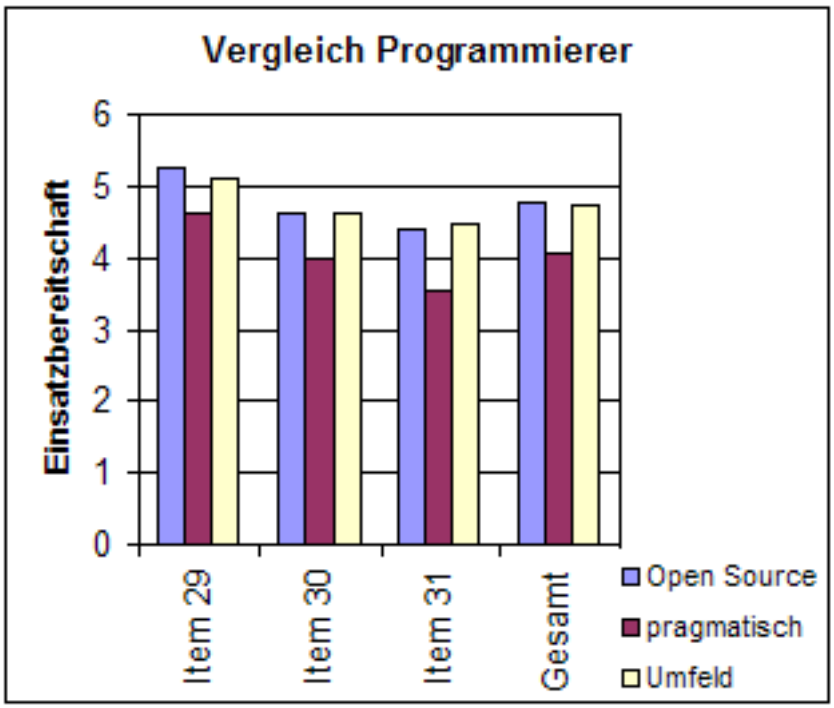


Abbildung 7.20: Einsatzbereitschaft von Programmierern

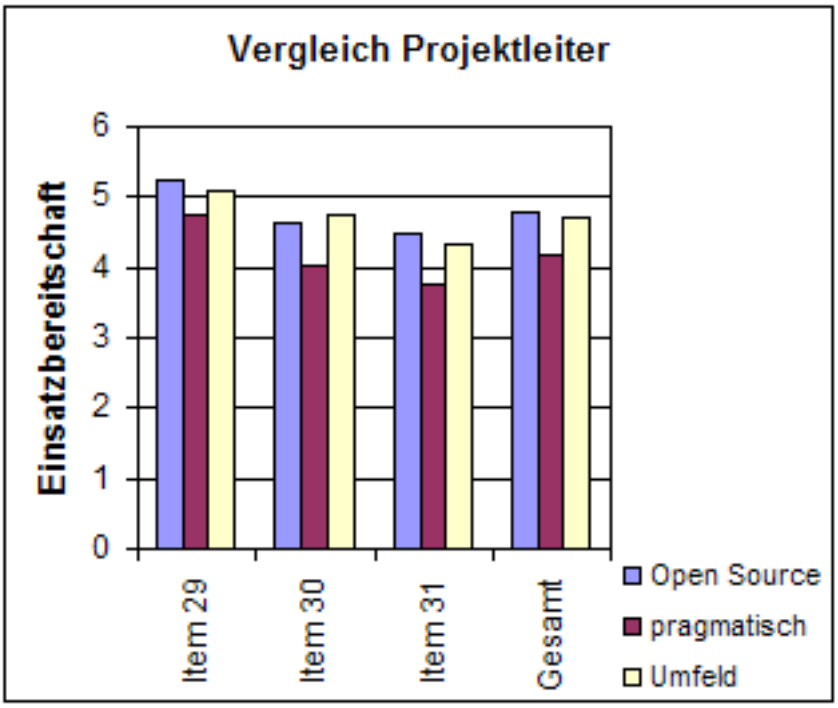


Abbildung 7.21: Einsatzbereitschaft von Projektleitern

Tabelle 7.64: Einsatzbereitschaft der unterschiedlichen Typen

Programmierer	**Einsatzbereitschaft (Gesamtindex)**		
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.76	0.04	523
pragmatischer Nutzen	4.07	0.07	259
soziales Umfeld	4.73	0.05	338
Gesamt	4.59	0.03	1120
Projektleiter	**Einsatzbereitschaft (Gesamtindex)**		
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.78	0.04	507
pragmatischer Nutzen	4.19	0.07	273
professionelles Umfeld	4.71	0.07	181
Gesamt	4.60	0.03	961

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

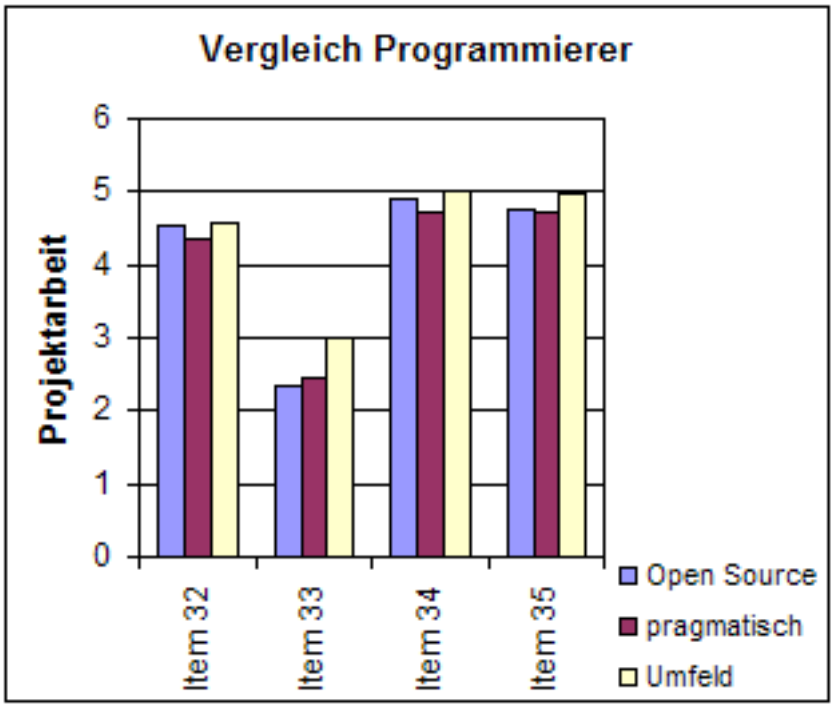


Abbildung 7.22: Einschätzung der Projektarbeit durch Programmierer

im Rahmen eines bezahlten Engagements an Open-Source-Projekten beteiligen. Dieses Ergebnis ist somit konsistent mit dem Befund aus Kapitel 7.4.3.

Die Wichtigkeit einer Projektvision (Item 34) wird vom pragmatisch motivierten Typ signifikant nüchterner eingestuft als von den anderen Typen ($\alpha = 1\%$, sowohl bei den Programmierern wie auch bei den Projektleitern), ebenso die effektive Häufigkeit solcher Visionen in den Open-Source-Projekten (Item 32, $\alpha = 5\%$). Allerdings gilt der letzte Befund nur für Programmierer. Bei den Projektleitern zeigen die unterschiedlichen Typen, was die Häufigkeit von Projektvisionen betrifft, keine Differenzen.

Signifikant verschieden ist auch die Einschätzung der Bedeutung der fachlichen Kompetenz des Projektleiters (Item 35). Bei den Programmierern sind es die durch das Umfeld motivierten Beitragsleister, welche dieser Kompetenz signifikant mehr Bedeutung zumessen ($\alpha = 1\%$), während sich die beiden anderen Typen in der Einschätzung teilen. Bei den Projektleitern ist es dagegen der pragmatische Typ, welcher durch eine signifikant nüchternere Einschätzung der Bedeutung von fachlicher Kompetenz heraussticht ($\alpha = 5\%$).

7.4.6 Weitere typenspezifische Mermale

Im Weiteren stellt sich die Frage, ob sich die unterschiedlichen Typen auch bezüglich des Outputs (Anzahl der Patches bzw. Klassen/Module/Files pro Jahr) unterscheiden. Allerdings zeigt eine Untersuchung der Mittelwerte in dieser Hinsicht

Tabelle 7.65: Einschätzung der Projektarbeit durch Programmierer

	Mittelwert	Std.-Fehler	Anzahl
32: Wie oft sind die Open-Source-Projekte, an denen Sie sich beteiligen, von einer klaren Projektvision getragen?			
Open-Source-Bewegung	4.55	0.05	514
pragmatischer Nutzen	4.34	0.08	256
soziales Umfeld	4.56	0.06	337
Gesamt	4.51	0.03	1107
33: Bei Ihrer Arbeit für Open-Source- Projekte: Wie häufig sind Abgabetermine spürbar?			
Open-Source-Bewegung	2.33	0.05	517
pragmatischer Nutzen	2.47	0.08	258
soziales Umfeld	2.99	0.07	337
Gesamt	2.56	0.04	1112
34: Wie wichtig ist die Vision hinter einem Open-Source-Projekt für Ihre Projektbeteiligung?			
Open-Source-Bewegung	4.92	0.05	519
pragmatischer Nutzen	4.72	0.08	260
soziales Umfeld	5.00	0.05	339
Gesamt	4.90	0.03	1118
35: Wie wichtig ist die fachliche Kompetenz des Projekteigentümers für Ihr Engagement an einem Open-Source-Projekt?			
Open-Source-Bewegung	4.77	0.05	512
pragmatischer Nutzen	4.72	0.08	257
soziales Umfeld	4.97	0.06	339
Gesamt	4.82	0.04	1108

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.66: Einschätzung der Projektarbeit durch Projektleiter

	Mittelwert	Std.-Fehler	Anzahl
32: Wie oft sind die Open-Source-Projekte, an denen Sie sich beteiligen, von einer klaren Projektvision getragen?			
Open-Source-Bewegung	4.6	0.05	506
pragmatischer Nutzen	4.47	0.07	268
professionelles Umfeld	4.45	0.08	181
Gesamt	4.53	0.04	955
33: Bei Ihrer Arbeit für Open-Source- Projekte: Wie häufig sind Abgabetermine spürbar?			
Open-Source-Bewegung	2.36	0.06	506
pragmatischer Nutzen	2.38	0.08	272
professionelles Umfeld	3.16	0.1	180
Gesamt	2.52	0.04	958
34: Wie wichtig ist die Vision hinter einem Open-Source-Projekt für Ihre Projektbeteiligung?			
Open-Source-Bewegung	4.96	0.05	504
pragmatischer Nutzen	4.71	0.08	275
professionelles Umfeld	5.06	0.08	180
Gesamt	4.91	0.04	959
35: Wie wichtig ist die fachliche Kompetenz des Projekteigentümers für Ihr Engagement an einem Open-Source-Projekt?			
Open-Source-Bewegung	4.9	0.05	496
pragmatischer Nutzen	4.71	0.08	271
professionelles Umfeld	4.93	0.08	178
Gesamt	4.85	0.04	945

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

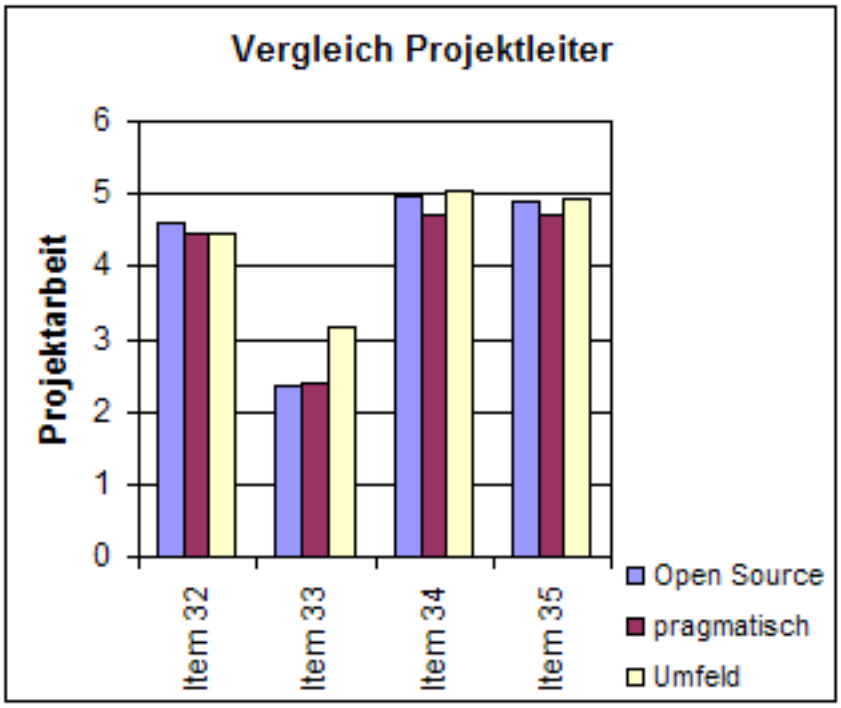


Abbildung 7.23: Einschätzung der Projektarbeit durch Projektleiter

keine signifikanten Unterschiede.

Hingegen kann eine interessante Differenz festgestellt werden, wenn die durchschnittliche der Anzahl an Jahren Erfahrung im Open-Source-Bereich der verschiedenen Typen untersucht wird. Hier weist der von Open Source faszinierte Typ mit 4.5 Jahren die geringste Erfahrung auf, während der pragmatisch motivierte Programmierer mit 5.3 Jahren signifikant mehr Erfahrung hat (α = 5%, siehe Tabelle 7.67). Dieses Resultat kann so gedeutet werden, dass die Beitragsleister mit zunehmender Erfahrung pragmatischer werden.

Insgesamt vermitteln diese Untersuchungen ein recht konsistentes Bild der drei Typen von Beitragsleistern, die aus der Analyse der Fragen zur ursprünglichen Teilnahmemotivation gewonnen wurden. Der durch die Open-Source-Bewegung beeinflusste Typ ist eher intrinsisch motiviert. Er ist im Untersuchungssample am besten vertreten (47% bei den Programmierern, 53% bei den Projektleitern). Dieser Typ vereinigt die von Lakhani/Wolf (in Lakhani und Wolf, 2003) identifizierten Cluster der intrinsisch motivierten Programmierer, die sich zur intellektuellen Stimulation sowie zur Verbesserung der Programmierfähigkeiten engagieren, sowie der Entwickler, die auf Grund von Community-basierten Normen beitragen. Der zweite Typ ist pragmatisch motiviert. Der hauptsächliche Anreiz für sein Engagement ist ein extrinsisches Bedürfnis nach Funktionalität. Der pragmatisch motivierte Typ kommt in dieser Weise auch in der Studie von Lakhani/Wolf vor. Der dritte Typ ist durch sein Umfeld zum Open-Source-Engagement motiviert worden. Bei den Projektleitern handelt es sich bei diesem Umfeld eindeutig um das professio-

Tabelle 7.67: Erfahrung der unterschiedlichen Typen

Programmierer	**Anzahl Jahre Erfahrung**		
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.53	0.19	522
pragmatischer Nutzen	5.32	0.34	262
soziales Umfeld	4.89	0.23	337
Gesamt	4.82	0.14	1121
Projektleiter	**Anzahl Jahre Erfahrung**		
	Mittelwert	Std.-Fehler	Anzahl
Open-Source-Bewegung	4.56	0.18	507
pragmatischer Nutzen	5.46	0.35	276
professionelles Umfeld	4.97	0.32	179
Gesamt	4.90	0.15	962

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

nelle Umfeld, d.h. es ist der Arbeitgeber, welcher hinter dem Engagement steht. Bei den Programmierern sind es auch die Kollegen aus dem sozialen Umfeld, welche den Beitragsleister zum Engagement motivieren. Auch in dieser Hinsicht besteht eine Übereinstimmung zur Studie von Lakhani/Wolf.

Die anschliessenden Untersuchungen konnten die Interpretation der Typen weitgehend bestätigen. Bezüglich des Spasses an der Tätigkeit und dem Empfinden von Flow weisen die pragmatisch motivierten Typen signifikant tiefere Werte auf, während die intrinsisch motivierten Typen hohe Werte zeigen. Das zeitliche Engagement der durch Open Source motivierten Beitragsleister findet vorwiegend in der Freizeit statt, während die durch das Umfeld motivierten Projektleiter konsequenterweise am Arbeitsplatz für das Open-Source-Projekt aktiv sind. Die Einsatzbereitschaft von pragmatisch motivierten Software-Entwicklern fällt signifikant nüchterner aus als diejenige der anderen Typen. Bezüglich der Abgabetermine vermelden die durch das Umfeld motivierten Entwickler deutlich höhere Werte, während sich die pragmatischen Programmierer signifikant nüchterner zu den Visionen hinter den Open-Source-Projekten äussern.

Der Umstand, dass die Open-Source-Typen bezüglich der Daten zu Output und Produktivität keine Differenzen zeigen, bestätigt den in Kapitel 7.3.2 erhaltenen Eindruck, dass die zur Messung dieser Grössen gewählten Masse in einer breiten Studie nicht zu aussagekräftigen Resultaten führen.

7.5 Die Bedeutung von Spass

In diesem Kapitel will ich die Bedeutung von Spass für das Open-Source-Engagement quantifizieren und damit meine zentrale Forschungsfrage 1 beantworten. Nach den Vorarbeiten in den letzten Kapiteln habe ich die Daten so präpariert, dass ich sie in mein Modell einsetzen kann. Im ersten Abschnitt rekapituliere ich noch einmal kurz mein Modell. In den anschliessenden Abschnitten verwende ich dieses Modell, um die Regressionen mit verschiedenen Operationalisierungen des Open-Source-Engagements zu berechnen. Im abschliessenden Abschnitt interpretiere ich die gewonnenen Resultate.

7.5.1 Das Modell

In Kapitel 2.1 habe ich eine Produktionsfunktion hergeleitet, welche das Engagement von Open-Source-Entwicklern als Output mit dem Spass an der Tätigkeit sowie den Opportunitätskosten der Zeit als Input verknüpft. In diesem Modell kommen Spass und die Opportunitätskosten als unabhängige Variablen in quadratischer Form vor. Geht der quadratische Term mit negativem Vorzeichen in die Gleichung ein, so entspricht das einem abnehmenden Grenzeffekt.

$$E = c + a_{1j} * S_j - a_{2j} * S_j^2 + b_1 * Z - b_2 * Z^2 + \epsilon_0 \tag{7.1}$$

wo
E: freiwilliges, unentgeltliches Engagement
S_j: Spass, operationalisiert als Flow-Komponenten j
Z: Freizeit
$a_1, a_2, b_1, b_2 > 0$
ϵ_0: Fehlerterm

In den Kapiteln 7.1, 7.2 und 7.3 habe ich die im Modell enthaltenen Variablen untersucht. Als Kriteriumsvariable kann ich, wie in Kapitel 7.2 ausgeführt, den Zeiteinsatz oder die Einsatzbereitschaft verwenden. In Kapitel 7.3 habe ich gezeigt, wie die 28 den Spass beim Programmieren betreffenden Items auf vier grundlegende Komponenten zurückgeführt werden können. Zur Berechnung des Modells kann ich nun die berechneten Werte für die Komponenten (aus der rotierten Komponentenmatrix) verwenden oder alternativ mit den zu den jeweiligen Komponenten gehörenden Items einen Mittelwert bilden und diesen ins Modell einsetzen. Die verfügbare Freizeit der Probanden habe ich in Tabelle 7.17 kalkuliert.

Bevor das Modell mit Hilfe einer Regressionsanalyse mit den erhobenen Daten getestet werden kann, muss ich überprüfen, ob die Voraussetzungen für eine OLS-Regression (Methode der kleinsten Quadrate) gegeben sind. Die Prämisse der Multikollinearität wurde anhand der partiellen Korrelationen der unabhängigen Variablen abgeklärt. Alle unabhängigen Variablen des Schätzmodells liegen mit ihren Inflationsfaktoren der Varianz weit unter dem Schwellenwert von 4. Diese Aussage gilt selbstverständlich mit Ausnahme der Variablen, die sowohl mit linearen wie auch quadratischen Termen vorkommen. In diesen Fällen besteht zwangsläufig eine hohe Korrelation.
Eine Untersuchung der standardisierten Residuen zeigte, dass erhebliche heteros-

Tabelle 7.68: Einsatzbereitschaft als Funktion von Spass 1

abhängige Variable	Einsatzbereitschaft
unabhängige Variablen	**geschätzte Koeffizienten**
Flow/Spass	1.132***
	(0.393)
Zeitempfinden/Immersion	0.681***
	(0.234)
Zeitempfinden/Immersion^2	0.174***
	(0.086)
Konzentration	0.397***
	(0.137)
Eindeutigkeit der Aufgabe	0.176**
	(0.061)
Eindeutigkeit der Aufgabe^2	-0.178***
	(-0.086)
Konstante	13.715***
R^2	0.244
Anzahl Fälle	930

Bemerkung: *** Signifikant auf 1% Niveau
** Signifikant auf 5% Niveau
In Klammern die standardisierten Beta-Koeffizienten

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

kedastische Fehlerterme vorliegen. Auf Grund dieses Sachverhalts werden in der Folge heteroskedastizitäts-konsistente Standardfehler-Schätzverfahren für die OLS-Regressionskoeffizienten verwendet.[9] Damit sind die Voraussetzungen gegeben, um für die weitere empirische Analyse die OLS-Methode zur Schätzung der multiplen Regression einzusetzen.

7.5.2 Einsatzbereitschaft als Funktion von Spass

In einem ersten Versuch wurde das Modell mit dem Index der Einsatzbereitschaft (Fragen 29, 30 und 31) als abhängige Variable berechnet. Als unabhängige Variablen wurden die Ergebnisse aus der Faktorenanalyse[10] sowie die Freizeit in quadratischer Form eingesetzt.

Die Tabelle 7.68 gibt die signifikanten Koeffizienten dieser Regression wieder. Das Resultat zeigt keinen signifikanten Beitrag der verfügbaren Freizeit. Quadratische Terme existieren nur für die Komponenten „Zeitempfinden/Immersion“ und „Eindeutigkeit der Aufgabe“, wobei letztere das erwartete negative Vorzeichen auf-

[9] Das Verfahren ist in Hayes und Cai (2004) beschrieben.

[10] Das Resultat der Faktorenanalyse wurden wie folgt verwendet: Mit den Items, welche die jeweiligen Komponenten bilden, wurde der Mittelwert gebildet und dessen Quadrat berechnet.

Tabelle 7.69: Einsatzbereitschaft als Funktion von Spass 2

abhängige Variable	Einsatzbereitschaft
unabhängige Variablen	**geschätzte Koeffizienten**
Flow/Spass (log-transformiert)	7.473***
	(0.415)
Zeitempfinden/Immersion (log-transformiert)	0.902***
	(0.085)
Konstante	1.079
R^2	0.218
Anzahl Fälle	1077

Bemerkung: *** Signifikant auf 1% Niveau
In Klammern die standardisierten Beta-Koeffizienten

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

weist. Der Erklärungsgehalt dieses Modells beträgt rund 24%.

Eine weitere Möglichkeit, einen sinkenden Grenznutzen zu modellieren, besteht darin, die unabhängigen Variablen logarithmisch zu transformieren (Gleichung 7.2).

$$E = c + a_j * \ln S_j + b * \ln Z + \epsilon_0 \quad (7.2)$$

wo
E: freiwilliges, unentgeltliches Engagement
S_j: Spass, Flow-Komponenten j
Z: Freizeit
$a, b > 0$
ϵ_0: Fehlerterm

Wie die Analyse zeigt, hat das logarithmische Modell praktisch die gleich grosse Aussagekraft wie das quadratische Modell (Tabelle 7.69). In diesem Modell hat der Faktor „Flow/Spass“ den grössten Anteil. Statt des Faktors „Eindeutigkeit der Aufgabe“ trägt nun der Faktor „Zeitempfinden/Immersion“ zur Einsatzbereitschaft bei. Dieses Modell kann rund 22% der Varianz der abhängigen Variable erklären.

Einen noch grösseren Erklärungsgehalt hat das Modell, wenn es mit den ursprünglichen (log-transformierten) Variablen statt den Faktoren als unabhängigen Variablen berechnet wird. Das Modell umfasst, nachdem auftretende Multikollinearitäten eliminiert worden sind, die sieben Aussagen „Meine Konzentration ist sehr hoch“ (20), „Ich freue mich schon vorher aufs Programmieren“ (21), „Die Handlung macht mir Spass“ (22), „Ich fühle mich überfordert“ (23), „Alles scheint wie von selbst zu laufen“ (24), „Ich führe die Handlung um ihrer selbst willen aus“ (26) und „Ich konzentriere mich voll aufs Programmieren“ (27). Interessant ist, dass vier der fünf Variablen des Faktors „Flow/Spass“ in diesem Modell relevant sind, zusätzlich zu zwei Variablen des Faktors „Konzentration“. Dies scheint allerdings konsistent mit den früheren Resultaten, in welchen die Flow-Komponente

Tabelle 7.70: Einsatzbereitschaft als Funktion von Spass und Zeit 3

abhängige Variable	Einsatzbereitschaft
unabhängige Variablen	**geschätzte Koeffizienten**
Item 21 (log-transf.) (Flow/Spass)	3.748***
	(0.286)
Item 22 (log-transf.) (Flow/Spass)	2.387***
	(0.135)
Item 26 (log-transf.) (Flow/Spass)	1.207***
	(0.103)
Item 27 (log-transf.) (Konzentration)	0.830***
	(0.094)
Item 24 (log-transf.) (Flow/Spass)	0.798***
	(0.104)
Item 20 (log-transf.) (Konzentration)	-1.015***
	(-0.093)
Item 23 (log-transf.)	-0.497***
	(-0.066)
Alter	0.-0.064***
	(-0.183)
R^2	0.271
Anzahl Fälle	1104

Bemerkung: *** Signifikant auf 1% Niveau
In Klammern die standardisierten Beta-Koeffizienten

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

jeweils mit dem grössten Koeffizienten auftauchte. Nicht den Erwartungen entspricht das negative Vorzeichen des Koeffizienten der Fragen 20 und 23.

Wird das Modell mit den demographischen Variablen kontrolliert, so zeigt sich, dass nur das Alter einen signifikanten Beitrag beisteuert, und zwar mit negativem Vorzeichen.[11] Je älter die Open-Source-Programmierer sind, desto geringer ist ihre Einsatzbereitschaft. Das R^2 dieses Modells beträgt 27% (Tabelle 7.70).

Zum Abschluss habe ich auch noch ein lineares Modell gerechnet. Diese Regression führt praktisch zum gleichen Ergebnis. Mit den sieben signifikanten Flow-Items und dem Alter können 27% der Varianz der abhängigen Variablen erklärt werden. Der einzige Unterschied ist, dass statt dem Item 23 in diesem Fall das Item 2 („Ich habe kein Gefühl mehr dafür, wie lange ich schon am Programmieren bin“), welches zur Flow-Komponente „Zeitempfinden/Immersion“ gehört, mit signifikantem Beitrag auftaucht (siehe Tabelle 7.71).

[11] Das Modell wurde mit folgenden Variablen kontrolliert: Alter, Anzahl der Jahre mit Engagement im Open-Source-Bereich, Geschlecht, Beschäftigungsgrad, Vorhandensein eines Kindes bzw. anderen Erwachsenen.

Tabelle 7.71: Einsatzbereitschaft als Funktion von Spass und Zeit 4

abhängige Variable	Einsatzbereitschaft
unabhängige Variablen	**geschätzte Koeffizienten**
Item 21 (Flow/Spass)	0.906***
	(0.284)
Item 22 (Flow/Spass)	0.442***
	(0.113)
Item 27 (Konzentration)	0.267***
	(0.090)
Item 26 (Flow/Spass)	0.261***
	(0.112)
Item 24 (Flow/Spass)	0.252***
	(0.094)
Item 2 (Zeitempfinden/Immersion)	0.146**
	(0.072)
Item 20 (Konzentration)	-0.280***
	(-0.097)
Alter	-0.060***
	(-0.170)
R^2	0.270
Anzahl Fälle	1088

Bemerkung: *** Signifikant auf 1% Niveau
Bemerkung: ** Signifikant auf 5% Niveau
In Klammern die standardisierten Beta-Koeffizienten

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 7.72: Engagement als Funktion von Spass und Zeit 1

abhängige Variable	Engagement in Freizeit
unabhängige Variablen	**geschätzte Koeffizienten**
Zeitempfinden/Immersion	1.082***
	(0.135)
Zeitempfinden/Immersion^2	0.486***
	(0.086)
Flow/Spass	0.980***
	(0.119)
Eindeutigkeit der Aufgabe	0.510**
	(0.063)
Konzentration	0.503**
	(0.062)
Freizeit	0.250***
	(0.778)
Freizeit^2	-0.002***
	(-0.408)
Konstante	8.264***
R^2	0.338
Anzahl Fälle	911

Bemerkung: *** Signifikant auf 1% Niveau
** Signifikant auf 5% Niveau
In Klammern die standardisierten Beta-Koeffizienten

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

7.5.3 Engagement als Funktion von Spass

In einer weiteren Analyse wurde das Modell mit dem zeitlichen Engagement (Anzahl der Stunden pro Woche für Open Source) in der Freizeit als abhängige Variable getestet.

In dieser Regression (siehe Tabelle 7.72) verschwinden die quadratischen Terme weitgehend. Nur noch die Opportunitätskosten der Zeit sowie die Komponente „Zeitempfinden/Immersion“ sind mit einem quadratischen Term enthalten, wobei der quadratische Term in der Zeit das erwartete negative Vorzeichen hat. Alle Terme sind stark signifikant und der Erklärungsgehalt dieser Regression beträgt 34%. Die vergleichsweise hohen Beta-Koeffizienten der Freizeit-Terme sind ein Hinweis darauf, dass das zeitliche Engagement hauptsächlich durch die verfügbare Freizeit der Open-Source-Programmierer bestimmt ist.

Ein Modell, in welchem das Engagement linear vom empfundenen Spass abhängt, entspricht nicht der üblichen ökonomischen Annahme eines sinkenden Grenznutzens. Aus diesem Grund wurde eine Regression gerechnet, in welcher die den Spass betreffenden Variablen logarithmisch transformiert in die Berechnung eingehen.

Tabelle 7.73: Engagement als Funktion von Spass und Zeit 2

abhängige Variable	Engagement in Freizeit
unabhängige Variablen	**geschätzte Koeffizienten**
Flow/Spass (log-transformiert)	3.850*
	(0.069)
Zeitempfinden/Immersion (log-transformiert)	2.860***
	(0.093)
Eindeutigkeit (log-transformiert)	2.418*
	(0.055)
Freizeit	0.374***
	(1.118)
Freizeit^2	-0.002***
	(-0.697)
Konstante	-12.980
R^2	0.310
Anzahl Fälle	977

Bemerkung: *** Signifikant auf 1% Niveau
** Signifikant auf 5% Niveau
* Signifikant auf 10% Niveau
In Klammern die standardisierten Beta-Koeffizienten

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Diese Modell mit dem gewünschten Verhalten ist, obwohl es eine Variable weniger beinhaltet, annähernd so erfolgreich wie das lineare Modell (siehe Tabelle 7.73). Allerdings sind in diesem Fall die Komponenten „Flow/Spass“ und „Eindeutigkeit der Aufgabe“ nur noch auf dem 10%-Niveau signifikant. Mit der verfügbaren Freizeit (geht quadratisch in das Modell ein) sowie dem Spass und dem Zeitempfinden (log-transformiert) können 31% des Phänomens erklärt werden.

7.5.4 Die Bedeutung von Spass bei Professionals

In der Tabelle 7.18 habe ich gezeigt, dass ein gewisser Anteil der Software-Entwickler, die an der Open-Source-Umfrage teilgenommen haben, als Professionals bezeichnet werden können, d.h. als Programmierer, die während ihrer Arbeitszeit an Open-Source-Projekten arbeiten. Welche Bedeutung hat die Freude am Programmieren für Software-Entwickler, die für ihre Arbeit bezahlt werden?

Die Tabelle 7.74 zeigt die signifikanten Koeffizienten der Regression der Flow-Komponenten mit der Einsatzbereitschaft der Professionals. Diese Regression besteht aus nur zwei Faktoren „Flow/Spass“ und „Zeitempfinden/Immersion“, die beide hochsignifikant beitragen und erstaunliche 44% der Varianz in der abhängigen Variablen erklären. Die Einsatzbereitschaft der Professionals ist demnach sehr stark geprägt von der Freude, welche das Programmieren ihnen bereitet.

Tabelle 7.74: Professionelle Einsatzbereitschaft als Funktion von Spass

abhängige Variable	Einsatzbereitschaft
unabhängige Variablen	**geschätzte Koeffizienten**
Flow/Spass	1.552***
	(0.491)
Zeitempfinden/Immersion	1.041***
	(0.337)
Konstante	12.958
R^2	0.436***
Anzahl Fälle	97

Bemerkung: *** Signifikant auf 1% Niveau
In Klammern die standardisierten Beta-Koeffizienten

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Als Professionals habe ich Open-Source-Entwickler definiert, die mindestens 90% ihres Engagements während der Arbeitszeit entrichten. Aus Tabelle 7.75 wird ersichtlich, dass der Freizeiteinsatz, den professionelle Open-Source-Entwickler leisten, ausschliesslich durch deren Verfügbarkeit von Freizeit bestimmt ist. Zwei Merkmale fallen bei diesem Resultat auf: 1.) Die Modellqualität ist bemerkenswert hoch, das Modell vermag 85% der Varianz in der abhängigen Variablen zu erklären. 2.) Der quadratische Term einer zusätzlichen Einheit Freizeit ist zwar klein, aber er hat ein positives Vorzeichen. Bei professionellen Open-Source-Entwicklern können wir demnach keinen sinkenden Effekt einer zusätzlichen Stunde Freizeit auf das Open-Source-Engagement feststellen.

Tabelle 7.75: Professioneller Freizeiteinsatz als Funktion von Spass

abhängige Variable	Engagement in Freizeit
unabhängige Variablen	**geschätzte Koeffizienten**
Freizeit	0.0501^{***}
	(0.739)
Freizeit^2	0.0003^{***}
	(0.253)
Konstante	1.392
R^2	0.853
Anzahl Fälle	99

Bemerkung: *** Signifikant auf 1% Niveau
In Klammern die standardisierten Beta-Koeffizienten

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

7.6 Interpretation

Die Regressionsrechnungen im Kapitel 7.5 erlauben folgende Interpretation:

1. Spass spielt eine Rolle: Ein einfaches Modell, welches Spass und Freizeit als erklärende Variablen enthält, kann zwischen 27% und 34% des Engagements für Open Source erklären.

2. Die Verfügbarkeit von Freizeit spielt keine Rolle, wenn die Open-Source-Entwickler nach der Einsatzbereitschaft, d.h. nach dem zukünftigen Engagement gefragt werden. Hingegen hat das Vorhandensein von Freizeit eine signifikante Bedeutung beim aktuellen Zeiteinsatz der Programmierer. Dieses nachvollziehbare Ergebnis kann als Hinweis für die Validität der Berechnungen gedeutet werden.

3. Die Freude am Programmieren nützt sich nicht ab: jede zusätzliche Einheit „Spass“ wird linear in zusätzliches Engagement umgesetzt. Dies kann gefolgert werden aus dem Sachverhalt, dass die Komponente „Flow/Spass“ nur mit linearem Term signifikant zur Regression beiträgt.

In Kapitel 7.1.4 habe ich das zeitliche Engagement der Open-Source-Entwickler untersucht und festgestellt, dass ein Open-Source-Programmierer im Durchschnitt 12.6 Stunden pro Woche an einem oder mehreren Open-Source-Projekten arbeitet. Von diesen 12.6 Stunden entfallen 7.3 Stunden auf die Freizeit des Programmierers. Damit entstehen 58% des Codes von Open-Source-Software in der Freizeit, während beachtliche 42% des Codes (5.2 Stunden pro Woche) auf Grund von bezahlter Tätigkeit zustande kommen. Wenn ich das Open-Source-Engagement der einzelnen Programmierer untersuche, so sehe ich, dass rund 66% mehrheitlich in der Freizeit an Open-Source-Projekten arbeiten, während rund ein Drittel der Software-Entwickler für ihre Arbeit an Open-Source-Projekten mehrheitlich (d.h.

mehr als 50% ihrer Zeit) bezahlt wird. Auf Grund dieser Daten lässt sich folgern, dass es sich bei Open Source (noch) um ein Freizeit-Phänomen handelt, dass aber der bezahlte Beitrag zu Open Source einen respektablen Anteil gewonnen hat.

Bei der Interpretation dieser Zahlen ist allerdings zu berücksichtigen, dass der professionelle Anteil Open-Source-Entwickler in meiner Untersuchung mit Sicherheit untervertreten ist. Die Software-Entwickler, die sich an meiner Untersuchung beteiligten, hatte ich über die Open-Source-Plattformen SourceForge, GNU\Savannah und BerliOS angesprochen. Diese Plattformen sind mit ihrer Infrastruktur sehr attraktiv für Freizeitprojekte. Professionellere Open-Source-Projekte vermögen allerdings ihre eigene Infrastruktur zu unterhalten und ziehen deshalb einen eigenen, unabhängigen Auftritt vor (z.B. Linux, Apache, Eclipse, Zope etc.). Sie signalisieren damit, dass sie einen Reifegrad erreicht haben und somit die Unterstützung durch beispielsweise SourceForge nicht mehr benötigen. Die Entwickler dieser Projekte konnte ich mit meiner Vorgehensweise nicht erreichen. Auf Grund dieses Sachverhalts vermute ich, dass der professionelle und bezahlte Anteil zu Open Source mit dem Beitrag, der in der Freizeit entsteht, gleichgezogen oder diesen mittlerweile schon überholt hat.

Die Resultate diese Auswertung zeigen, dass es mit der in dieser Untersuchung vorgestellten Methode möglich ist, die Bedeutung des Motivationsfaktors „Spass“ zu quantifizieren. Sowohl das Engagement für Open Source (in Form des Zeitaufwands oder der Einsatzbereitschaft) wie auch der bei diesem Engagement erlebte Spass können bestimmt werden. In meiner Studie habe ich ein einfaches Modell verwendet, um diese beide Grössen miteinander in Beziehung zu setzen.

Mit diesem Modell war es mir möglich, einen beachtlichen Anteil des Engagements für Open Source zu erklären. Das Ergebnis zeigt aber auch, dass mit Spass, zumindest in dieser einfachen Form, nicht alles begründet werden kann. Dieses Resultat ist damit in Übereinstimmung mit den Vermutungen, dass beim Open-Source-Engagement eine Vielzahl von Motivationen miteinander koexistieren können (siehe z.B. Franck und Jungwirth (2002b) oder Lakhani und Wolf (2003)). Interessant wäre es, wenn mit analogen Methoden auch die Bedeutung anderer Motivationsfaktoren wie z.B. Reputation oder Altruismus quantitativ bestimmt werden könnte.

In meiner Untersuchung zeige ich einerseits, dass Spass ein wichtiger Faktor ist, um das unbezahlte Engagement der Open-Source-Entwickler zu erklären. Andererseits vermute ich, dass ein immer grösserer Anteil Open Source professionell erstellt wird. Welche Perspektiven für den Motivationsfaktor „Spass“ lassen sich aus dieser Entwicklung ableiten?

Damit ein Programmierer in seiner Freizeit an Open-Source-Projekten arbeitet, braucht er Freizeit und Spass am Programmieren. Diese Voraussetzungen sind nachvollziehbar und werden durch meine Daten eindrücklich bestätigt. Ebenso klar ist, dass das Engagement eines bezahlten Open-Source-Entwicklers durch andere Faktoren als seine Freude am Entwickeln von Software bestimmt wird. Dementsprechend würde ein Vordringen des bezahlten Engagements im Open-Source-Bereich bedeuten, dass der Faktor „Spass“ relativ an Bedeutung verliert.

Diese Folgerung ist allerdings nur stimmig, wenn wir das Engagement in Form des zeitlichen Einsatzes für Open Source untersuchen. Wenn wir dagegen die Einsatzbereitschaft und deren Beziehung zu Spass betrachten, so sieht das

Bild anders aus. Im Falle der Open-Source-Entwickler allgemein vermögen die Flow-Komponenten rund 27% der Varianz der Einsatzbereitschaft zu erklären. Bei professionellen Open-Source-Entwicklern steigt dieser Wert auf 44%. Demnach kommt dem Faktor „Spass“, was die Einsatzbereitschaft betrifft, im professionellen Open-Source-Umfeld eine erhöhte Bedeutung zu.

Zu vermuten ist, dass Spass über die gesteigerte Einsatzbereitschaft einen positiven Effekt auf die Produktivität der Software-Entwickler hat. Sollte diese Vermutung zutreffen, so würde die Bedeutung von Spass bei wachsender Professionalisierung nicht gemindert, sonder eher noch zunehmen.

In meiner Untersuchung habe ich versucht, die Produktivität der Programmierer zu messen. Mit den Daten, die ich mit meiner Operationalisierung von Produktivität gewonnen habe, kann ich aber nur rund 1% der Varianz des Engagements erklären. Dieses wenig aussagekräftige Resultat habe ich damit erklärt, dass die von mir gewählte Art, Produktivität zu operationalisieren, dem Sachverhalt nicht angemessen ist. Ich werde im folgenden Kapitel nochmals auf das Problem, wie die Produktivität von Software-Entwicklern gemessen werde kann, eingehen.

An dieser Stelle will ich den Stand meiner Auswertungen nochmals rekapitulieren. Ich habe meine ersten zwei Forschungsfragen beantworten können. Spass spielt eine Rolle für Open Source: Mit diesem Motivationsfaktor können zwischen 27% und 33% des Engagements der Open-Source-Entwickler erklärt werden (Forschungsfrage 1). Aus meinen Daten lässt sich ablesen, dass sich das Open-Source-Phänomen mehrheitlich in der Freizeit abspielt (Forschungsfrage 2). Auf Grund der Auswahl der Probanden für meine Untersuchung muss ich diesen Befund allerdings relativieren. Ich vermute statt dessen, dass das bezahlte Engagement für Open Source mit dem Freizeitengagement gleichgezogen hat. Nun zeigen meine Daten, dass für bezahlte Open-Source-Programmierer Spass an Bedeutung verliert, wenn es um den Zeiteinsatz der Software-Entwickler geht. Es wäre allerdings falsch, aus diesem Resultat den Schluss zu ziehen, dass Spass im bezahlten Umfeld nicht mehr wichtig ist. Im Gegenteil: bezüglich der Einsatzbereitschaft wird die Bedeutung von Spass für bezahlte Open-Source-Entwickler akzentuiert. Aus diesem Grund lässt sich folgern, dass Spass für Open Source eine wichtige Rolle spielt, ganz unabhängig davon, ob die Open-Source-Entwickler für ihr Engagement bezahlt werden oder nicht.

Kapitel 8

Befragung der kommerziellen Software-Entwickler

8.1 Deskriptive Auswertungen

Mit diesem Kapitel beginne ich die Untersuchung der Umfrage unter den Software-Entwicklern aus dem kommerziellen Bereich. Wie die Open-Source-Umfrage will ich auch diesen Teil mit einigen deskriptiven Auswertungen einleiten, um auf diese Weise einen allgemeinen Eindruck über das Sample zu vermitteln.

8.1.1 Allgemeines Antwortverhalten

Der FASD-Fragebogen für die Software-Entwickler im kommerziellen Umfeld wurde von 114 Software-Entwicklern ausgefüllt. Diese Anzahl ist, um repräsentative Aussagen machen zu können, ungenügend. Die Aussagen, die ich auf Grund der Auswertung dieser Daten machen werde, stehen deshalb unter dem Vorbehalt mangelnder Repräsentativität.

Für diese Umfrage konnten sechs Schweizer Software-Firmen gewonnen werden, welche die Anfrage zur Teilnahme an der FASD-Studie an insgesamt 310 Programmierer weiterleiteten. Beim kommerziellen Teil der Studie konnte somit eine Rücklaufquote von 37% erreicht werden.

Die Tabelle 8.1 zeigt, dass die kommerzielle Version des Fragebogens von 68% der Probanden vollständig ausgefüllt wurde. Bei der Open-Source-Umfrage betrug dieser Anteil 48%. Die grösste Anzahl unbeantworteter Fragen in einem Fragebogen beträgt 12. Verglichen mit der Open-Source-Umfrage (siehe Tabelle 7.1) waren die kommerziellen Software-Entwickler demnach deutlich sorgfältiger beim Ausfüllen des Fragebogens.

Die Software-Entwickler in den angesprochenen Schweizer Firmen bevorzugten eindeutig die deutschsprachige Version des Fragebogens (94%). Die englische Version wurde nur von sieben Programmierern gewählt.

Die kommerziellen Software-Entwickler benötigten im Durchschnitt 8.8 Minuten, um den Fragebogen auszufüllen (Open-Source-Version: 8.9 Minuten).

Tabelle 8.1: Fehlende Werte

Anzahl fehlend	Anzahl Fälle	Prozent	Kumulierte Prozent
0	78	68.4%	68.4%
1	17	14.9%	83.3%
2	7	6.1%	89.5%
3	5	4.4%	93.9%
4	3	2.6%	96.5%
5	2	1.8%	98.2%
7	1	0.9%	99.1%
12	1	0.9%	100.0%
Total	114	100%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.2: Alter

		Anzahl	Prozent	Gültige Prozent
Gültig	10-19	5	4.4%	4.4%
	20-29	37	32.5%	32.7%
	30-39	48	42.1%	42.5%
	40-49	19	16.7%	16.8%
	50-59	3	2.6%	2.7%
	≥ 60	1	0.9%	0.9%
	Total	113	99.1%	100%
Fehlend		1	0.9%	
Total		114	100%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

8.1.2 Demographische Angaben

Die kommerziellen Software-Entwickler sind mehrheitlich zwischen 30 und 40 Jahre alt (Mittelwert: 33.7 Jahre, Median: 33 Jahre), wobei ein bedeutender Anteil (rund 33%) zwischen 20 und 30 Jahre alt ist (Tabelle 8.2). Sie haben im Durchschnitt 14 Jahre Programmiererfahrung (Median: 14 Jahre, vgl. Tabelle 8.3).

Verglichen mit dem Open-Source-Bereich (Frauenanteil 2%) ist der Frauenanteil unter kommerziellen Bedingungen mit 12% deutlich höher (Tabelle 8.4). Wie bei den Open-Source-Entwicklern haben auch im kommerziellen Bereich die Männer mehr Programmiererfahrung (fast vier Jahre, siehe Tabelle 8.5). Allerdings ist diese Differenz nur auf dem 10%-Niveau signifikant. Die Software-Entwickler, welche den Fragebogen ausgefüllt haben, sind zum überwiegenden Teil vollbeschäftigt (86%, Tabelle 8.6).

Tabelle 8.3: Anzahl Jahre Programmiererfahrung

	Anzahl	**Prozent**
0 - 5	27	23.7%
6 - 10	20	17.5%
11 - 15	20	17.5%
16 - 20	29	25.4%
21 - 25	14	12.3%
>25	4	3.5%
Total	114	100%

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.4: Geschlecht

		Anzahl	**Prozent**	**Gültige Prozent**
Gültig	männlich	99	86.8%	87.6%
	weiblich	14	12.3%	12.4%
	Total	113	99.1%	100%
Fehlend		1	0.9%	
Total		114	100%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.5: Programmiererfahrung nach Geschlecht

	Anzahl Jahre		
	Mittelwert	**Std.fehler**	**Anzahl**
männlich	13.87	0.73	99
weiblich	10.07	2.43	14
Total	13.39	0.70	114

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.6: Beschäftigungsgrad

		Anzahl	Prozent	Gültige Prozent
Gültig	100% beschäftigt	97	85.1%	85.8%
	90% - 99% beschäftigt	3	2.6%	2.7%
	80% - 89% beschäftigt	9	7.9%	8.0%
	70% - 79% beschäftigt	2	1.8%	1.8%
	60% - 69% beschäftigt	1	0.9%	0.9%
	30% - 39% beschäftigt	1	0.9%	0.9%
	Total	113	99.1%	100%
Fehlend		1	0.9%	
Total		114	100%	

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

8.1.3 Arbeitsplatz und Projektarbeit

Unter den 28 Item, welche das Flow-Erleben abfragen, erhält die Aussage 19 („Ich fühle mich der Aufgabe gewachsen“) mit 4.93 Punkten die höchste Zustimmung. Dies ist gleichzeitig das Item mit der kleinsten Varianz. Das Item 26 („Ich führe die Handlung um ihrer selbst willen aus“), welches mit 3.23 den geringsten Wert aufweist, zeichnet sich gleichzeitig durch eine der grössten Varianzen aus.

Tabelle 8.7: Flow-Erleben

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
1: Ich vergesse ganz die Zeit.	4.24	1.00	114
2: Ich habe kein Gefühl mehr dafür, wie lange ich schon am Programmieren bin.	3.79	1.30	112
3: Die Handlung erfolgt wie aus einem Guss.	4.03	1.00	111
4: Ich vergesse meine Sorgen.	3.96	1.37	113
5: Es fällt mir leicht, mich zu konzentrieren.	4.42	0.99	112
6: Ich gehe ganz in der Handlung auf.	4.35	1.08	113
7: Meine Aufmerksamkeit ist völlig aufs Programmieren gelenkt.	4.38	1.12	114
8: In der Situation sind die Anforderungen an mich völlig klar.	3.94	1.11	113
9: An Vergangenes oder Zukünftiges denke ich kaum.	3.51	1.32	112
10: Die Forderungen an mich sind eindeutig.	3.73	1.13	113
11: Es gibt viele Dinge, die ich lieber tun würde. (-)	3.55	1.30	114
12: Ich habe das Gefühl, die Anforderungen der Situation gut bewältigen zu können.	4.68	0.80	114

Tabelle 8.7: Flow-Erleben (Forts.)

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
13: Ich handle nur, weil ich weiss, dass es sich für mich bezahlt machen wird. (-)	4.32	1.15	110
14: Ich weiss immer genau, was zu tun ist.	3.84	1.02	114
15: Ich bin mit meinen Gedanken ganz woanders. (-)	4.77	0.90	111
16: Ich muss nicht über andere Dinge grübeln.	3.77	1.25	110
17: Mir ist klar, wie ich vorzugehen habe.	4.39	0.88	113
18: Ich bin voll und ganz bei der Sache.	4.57	0.85	114
19: Ich fühle mich der Aufgabe gewachsen.	4.93	0.71	114
20: Meine Konzentration ist sehr hoch.	4.77	0.80	114
21: Ich freue mich schon vorher aufs Programmieren.	4.63	0.99	113
22: Die Handlung macht mir Spass.	4.69	0.86	113
23: Ich fühle mich überfordert. (-)	4.79	1.01	114
24: Alles scheint wie von selbst zu laufen.	3.61	1.06	113
25: Ich vergesse alles um mich herum.	3.69	1.20	114
26: Ich führe die Handlung um ihrer selbst willen aus.	3.23	1.28	103
27: Ich konzentriere mich voll aufs Programmieren.	4.58	0.81	113
28: Ich lasse mich von anderen Dingen ablenken. (-)	3.98	1.14	114

Bei den Fragen zum Engagement am Arbeitsplatz erhält das Item 30 „Meine zukünftige Karriere plane ich im IT-Bereich“ die höchste Zustimmung. Die Zustimmungsrate von 4.7 unterscheidet sich hochsignifikant von den Werten der beiden anderen Items. Dagegen unterscheiden sich die Mittelwerte der Items 29 und 30 nicht signifikant (Tabelle 8.8).

Von den vier Fragen, welche das Verhältnis zum Arbeitgeber behandeln, weist das Item 33 den höchsten Wert auf. Allerdings unterscheidet es sich dabei nicht signifikant vom Item 32. Die geringste Zustimmung findet das Item 35. Dessen Mittelwert unterscheidet sich hochsignifikant von den anderen (Tabelle 8.9)[1].

Die befragten Software-Entwickler im kommerziellen Umfeld arbeiten praktisch nie an Open-Source-Projekten. Hingegen ist Open-Source-Software durchaus verbreitet am Arbeitsplatz von kommerziellen Software-Entwicklern (Tabelle 8.10).

Die Tabelle 8.11 zeigt die Zustimmungsraten zu den Aussagen bezüglich der Projektsituation. Die grösste Zustimmung finden die Items, in denen nach der Häufigkeit von Abgabeterminen, der Wichtigkeit einer Projektvision und der fach-

[1] Werte von 1=„nie“ bis 6=„immer“

Tabelle 8.8: Engagement am Arbeitsplatz

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
29: Am Wochenanfang freue ich mich auf die bevorstehende Arbeit.	4.09	0.93	114
30: Für meine Arbeit Überstunden zu machen, macht mir nichts aus.	4.16	1.28	113
31: Meine zukünftige Karriere plane ich im IT-Bereich.	4.67	1.08	110

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.9: Verhältnis zum Arbeitgeber

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
32: Ich bin stolz darauf, bei der Firma zu arbeiten.	4.01	1.36	111
33: Ich kann mich bei der Firma persönlich und beruflich weiterentwickeln.	4.15	1.36	112
34: Die Firma ist richtig aufgestellt und verfolgt die richtigen Ziele.	3.46	1.43	112
35: Bei der Firma werden das Rollenmodell und die Prozesse professionell und erfolgreich umgesetzt.	3.02	1.22	110

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.10: Open Source am Arbeitsplatz

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
36: In meiner Arbeitszeit arbeite ich an Open-Source-Projekten.	1.66	1.11	111
37: An meinem Arbeitsplatz verwende ich Open-Source-Software.	3.91	1.70	112

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.11: Projektsituation

	Mittel-Wert	Standard-Abweich.	Gültige Anzahl
38: An Ihrem Arbeitsplatz: Wie häufig sind Abgabetermine spürbar?	4.78	1.02	113
39: An Ihrem Arbeitsplatz: Wie oft sind die Software-Projekte von einer klaren Projektvision getragen?	3.67	1.24	109
40: An Ihrem Arbeitsplatz: Wie häufig ist die formale Autorität des Projektmanagers spürbar?	3.23	1.24	112
41: An Ihrem Arbeitsplatz: Wie wichtig ist eine Vision hinter einem Software-Projekt für ihre Arbeit als Software-Entwickler?	4.70	1.04	113
42: An Ihrem Arbeitsplatz: Wie wichtig ist die fachliche Kompetenz des Projektmanagers für Ihre Arbeit in einem Software-Projekt?	4.87	1.09	113

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

licher Kompetenz der Projektmanagers gefragt wird. Diese Items weisen untereinander keine signifikante Differenz auf. Die geringste Zustimmung findet die Aussage „Wie häufig ist die formale Autorität des Projektmanagers spürbar?“ (siehe Tabelle 8.11). Bemerkenswert ist die hochsignifikante Differenz zwischen Item 39, in welchem nach der Häufigkeit einer Projektvision gefragt wird, und Item 41, wo die Wichtigkeit einer Projektvision beurteilt wird. Offensichtlich besteht eine bedeutende Diskrepanz zwischen der aktuellen und der gewünschten Häufigkeit von Projektvisionen.

8.2 Flow und Engagement bei kommerziellen Programmierern

Den Software-Entwicklern im kommerziellen Umfeld wurden im ersten Teil des Fragebogens bezüglich des Flow-Empfindens die gleichen 28 Frage-Items zur Beantwortung vorgelegt wie den Open-Source-Entwicklern. Damit wird es möglich, die Antworten der Open-Source- mit denjenigen der kommerziellen Software-Entwickler zu vergleichen.

8.2.1 Faktorenanalyse der Flow-Items

Voraussetzungen für eine Faktorenanalyse

Wiederum will ich zuerst die 28 Items mit Hilfe einer Faktorenanalyse auf zugrunde liegende Faktoren untersuchen. Ein vorbereitender Test zeigt, dass der KMO-Wert für die Faktoren-Analyse mit 0.792 annähernd gut ist, während der Bartlett-Test auf Sphärizität einen hochsignifikanten Wert aufweist (Signifikanzniveau $\alpha < 1\%$). Die Anti-Image-Korrelationsmatrix weist allerdings in der Diagonale für die Frage-Items 13, 15, 16 und 26[2] schlechte Werte auf, weshalb diese Items aus der weiteren Analyse ausgeschlossen werden. Mit dieser Reduktion verbessert sich der KMO-Wert deutlich auf 0.847, womit die Faktorenanalyse als gut (*meritorious*) qualifiziert ist.

Die Faktorenanalyse mit den verbleibenden 24 Fragen legt eine Lösung mit fünf Faktoren nahe: der Screeplot der Analyse (siehe Abbildung 8.1) zeigt, wie nach dem fünften Faktor der Eigenwert kleiner als 1 wird. Der Erklärungsgehalt dieser fünf Faktoren beträgt 64.5% des ursprünglichen Gehalts.

Interpretation der Lösung

Die Tabelle mit den Kommunalitäten (Tabelle 8.12) zeigt, dass der Erklärungsgehalt der Lösung mit fünf Faktoren für die einzelnen Fragen zwischen 41% (Frage 3) und 74% (Frage 18) liegt.

Die Tabelle 8.13 mit der Interpretation der rotierten Komponenten zeigt eine recht gute Übereinstimmung mit der Faktorenanalyse der Antworten der Open-Source-Programmierer (Tabelle 7.37). Die Faktoren „Flow/Spass“, „Konzentration“, „Zeitempfinden/Immersion“ und „Eindeutigkeit der Aufgabe“ tauchen auch in dieser Analyse mit praktisch den gleichen Frage-Items auf.

Interessant ist, dass verglichen mit der Analyse der Antworten aus dem Open-Source-Umfeld die Faktoren „Aufmerksamkeit“ und „umgekehrte Bewertung“ verschwunden sind, dagegen erscheint neu der Faktor „Herausforderung“. Wenn wir uns allerdings erinnern, dass die beiden in dieser Auswertung nicht aufgetauchten Faktoren bezüglich der Reliabilität klar ungenügende Werte aufgewiesen haben (Cronbachs $\alpha < 0.41$), so erscheint diese Bild nicht mehr überraschend. Tatsächlich

[2] 13: „Ich handle nur, weil ich weiss, dass es sich für mich bezahlt machen wird“, 15: „Ich bin mit meinen Gedanken ganz woanders“, 16: „Ich muss nicht über andere Dinge grübeln“, 26: „Ich führe die Handlung um ihrer selbst willen aus“.

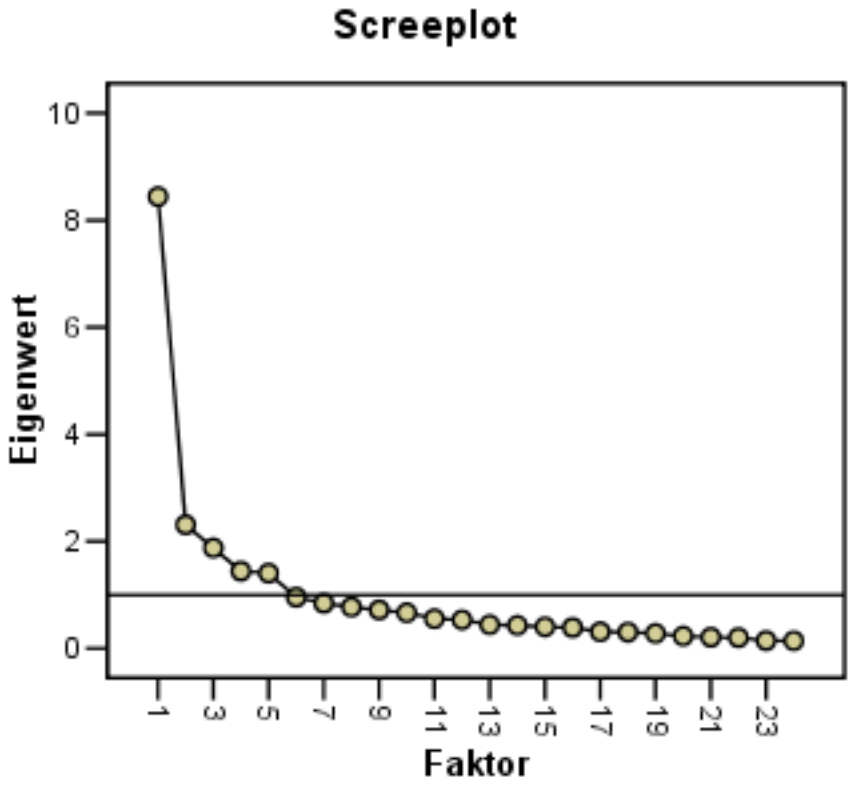


Abbildung 8.1: Screeplot der Faktorenanalyse

besitzen die in dieser Analyse gewonne Faktoren bezüglich der Reliabilität allesamt zufrieden stellende Werte. Mit 0.74 weist der Faktor „Eindeutigkeit der Aufgabe“ den kleinsten Wert für Cronbachs α auf.

Der neue Faktor „Herausforderung“ wird aus zwei Frage-Items gebildet, die in der ersten Befragung zum Faktor „Eindeutigkeit der Aufgabe“ gehörten. Zusätzlich kommt ein Item dazu, welches in der ersten Analyse auf Grund des schwachen KMO-Werts eliminiert worden ist. Die Tatsache, dass die Faktorenanalyse der Antworten von kommerziellen Software-Entwickler diesen Faktor „Herausforderung“ nahe legt, kann als Hinweis gedeutet werden, dass Herausforderung im Alltag von kommerziellen Programmierern eine wichtigere Rolle spielt als im Kontext eines Open-Source-Projekts.

Tabelle 8.12: Kommunalitäten der rotierten Lösung

1	Ich vergesse ganz die Zeit.	70.1%
2	Ich habe kein Gefühl mehr dafür, wie lange ich schon am Programmieren bin.	73.7%
3	Die Handlung erfolgt wie aus einem Guss.	41.1%

Tabelle 8.12: Kommunalitäten der rotierten Lösung (Forts.)

4	Ich vergesse meine Sorgen.	44.1%
5	Es fällt mir leicht, mich zu konzentrieren.	68.0%
6	Ich gehe ganz in der Handlung auf.	66.4%
7	Meine Aufmerksamkeit ist völlig aufs Programmieren gelenkt.	69.2%
8	In der Situation sind die Anforderungen an mich völlig klar.	55.0%
9	An Vergangenes oder Zukünftiges denke ich kaum.	67.2%
10	Die Forderungen an mich sind eindeutig.	68.8%
11	Es gibt viele Dinge, die ich lieber tun würde. (-)	53.5%
12	Ich habe das Gefühl, die Anforderungen der Situation gut bewältigen zu können.	73.6%
14	Ich weiss immer genau, was zu tun ist.	58.0%
17	Mir ist klar, wie ich vorzugehen habe.	60.8%
18	Ich bin voll und ganz bei der Sache.	74.1%
19	Ich fühle mich der Aufgabe gewachsen.	68.2%
20	Meine Konzentration ist sehr hoch.	69.1%
21	Ich freue mich schon vorher aufs Programmieren.	69.3%
22	Die Handlung macht mir Spass.	71.1%
23	Ich fühle mich überfordert. (-)	71.3%
24	Alles scheint wie von selbst zu laufen.	57.6%
25	Ich vergesse alles um mich herum.	72.1%
27	Ich konzentriere mich voll aufs Programmieren.	62.4%
28	Ich lasse mich von anderen Dingen ablenken. (-)	63.1%

Tabelle 8.13: Interpretation der Faktoren

Faktoren		**Ladung**
Flow/Spass		
Cronbachs α: 0.82, mittlere Inter-Item Korr.: 0.44		
3	Die Handlung erfolgt wie aus einem Guss.	0.402
6	Ich gehe ganz in der Handlung auf.	0.671
11	Es gibt viele Dinge, die ich lieber tun würde. (-)	0.695
21	Ich freue mich schon vorher aufs Programmieren.	0.814
22	Die Handlung macht mir Spass.	0.810
24	Alles scheint wie von selbst zu laufen.	0.429
Konzentration		
Cronbachs α: 0.86, mittlere Inter-Item Korr.: 0.52		
5	Es fällt mir leicht, mich zu konzentrieren.	0.751
7	Meine Aufmerksamkeit ist völlig aufs Programmieren gelenkt.	0.497
18	Ich bin voll und ganz bei der Sache.	0.689
20	Meine Konzentration ist sehr hoch.	0.717

Tabelle 8.13: Interpretation der Faktoren (Forts.)

Faktoren		Ladung
27	Ich konzentriere mich voll aufs Programmieren.	0.500
28	Ich lasse mich von anderen Dingen ablenken. (-)	0.778
Zeitempfinden/Immersion		
Cronbachs α: 0.77, mittlere Inter-Item Korr.: 0.42		
1	Ich vergesse ganz die Zeit.	0.784
2	Ich habe kein Gefühl mehr dafür, wie lange ich schon am Programmieren bin.	0.847
4	Ich vergesse meine Sorgen.	0.470
9	An Vergangenes oder Zukünftiges denke ich kaum.	0.565
25	Ich vergesse alles um mich herum.	0.655
Eindeutigkeit der Aufgabe		
Cronbachs α: 0.74, mittlere Inter-Item Korr.: 0.43		
8	In der Situation sind die Anforderungen an mich völlig klar.	0.644
10	Die Forderungen an mich sind eindeutig.	0.797
14	Ich weiss immer genau, was zu tun ist.	0.693
17	Mir ist klar, wie ich vorzugehen habe.	0.528
Herausforderung		
Cronbachs α: 0.79, mittlere Inter-Item Korr.: 0.58		
12	Ich habe das Gefühl, die Anforderungen der Situation gut bewältigen zu können.	0.783
19	Ich fühle mich der Aufgabe gewachsen.	0.765
23	Ich fühle mich überfordert. (-)	0.810

8.2.2 Spass und Engagement am Arbeitsplatz

Bei den professionellen Open-Source-Entwicklern habe ich gesehen, dass das zeitliche Engagement keinen Zusammenhang mit dem Flow-Empfinden der Programmierer hat, sondern von anderen Faktoren bestimmt ist (siehe Kapitel 7.5.4). Dieser Sachverhalt dürfte für Software-Entwickler, die an kommerziellen Projekten arbeiten, nicht weniger gültig sein. Dagegen konnte ich bei den professionellen Software-Entwicklern einen deutlichen Zusammenhang zwischen Einsatzbereitschaft und den Komponenten „Flow/Spass“ und „Zeitempfinden/Immersion“ feststellen. Kann eine solche Beziehung auch für Software-Entwickler im kommerziellen Bereich gefunden werden?

Als abhängige Variable für meine Regressionsanalyse berechne ich aus den Items 29, 30 und 31 einen Index „Engagement am Arbeitsplatz“ (siehe Tabelle 8.8). Als unabhängige Variablen gebe ich die im ersten Teil dieses Kapitels berechneten Werte der Flow-Komponenten in quadratischer Form ein. Tabelle 8.14 zeigt die signifikanten Koeffizienten dieser Regressionsanalyse. Wie bei der Regression gegen die Einsatzbereitschaft im Falle der Open-Source-Umfrage verschwinden alle quadratischen Terme. Übrig bleiben die beiden Komponenten „Flow/Spass“ und

Tabelle 8.14: Einsatzbereitschaft als Funktion von Spass

abhängige Variable	Engagement am Arbeitsplatz
unabhängige Variablen	**geschätzte Koeffizienten**
Flow/Spass	0.303***
	(0.428)
Herausforderung	0.144**
	(0.203)
Konstante	4.342
R^2	0.224
Anzahl Fälle	99

Bemerkung: *** Signifikant auf 1% Niveau
** Signifikant auf 5% Niveau
In Klammern die standardisierten Beta-Koeffizienten

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

„Herausforderung“ mit signifikanten Beiträgen zu einem Modell, welches 22% der Varianz in der abhängigen Variablen erklären kann. Diese Analyse macht deutlich, dass Spass (und Herausforderung) auch unter kommerziellen Bedingungen eine Rolle spielt.

8.3 Korrelationen

Werden die Flow-Faktoren mit den Werten für das Engagement am Arbeitsplatz, für das Verhältnis zum Arbeitgeber und anderen Werten korreliert, so ergeben sich einige interessante und signifikante Zusammenhänge, die es wert sind, genauer untersucht zu werden.

8.3.1 Bedingungen für Flow-Erleben

Die Tabelle 8.15 zeigt die signifikanten Korrelationen der Flow-Komponenten mit den Antworten zum Engagement am Arbeitsplatz (Fragen 29 bis 31). Es überrascht kaum, dass die Freude auf die bevorstehende Arbeit mit allen Flow-Komponenten signifikant korreliert. Der stärkste Zusammenhang ergibt sich mit der Komponente „Flow/Spass“. Auch die Bereitschaft, Überstunden zu machen, korreliert signifikant mit allen Flow-Komponenten. Schwächer ist dagegen die Korrelation der Karriereplanung im IT-Bereich mit den Flow-Komponenten. In diesem Fall gibt es nur noch für die Komponenten „Flow/Spass“ und „Herausforderung“ einen signifikanten Zusammenhang. Es scheint aber konsistent, dass der Faktor „Herausforderung“ in diesem Punkt bedeutender ist. Für die Karriereplanung ist es entscheidend, welche Herausforderung die Arbeit bietet.

In Tabelle 8.16 sind die Korrelationen der Flow-Komponenten mit den Fragen betreffend das Verhältnis zum Arbeitgeber (Fragen 32 bis 35) aufgeführt. Die Flow-Komponenten „Flow/Spass“ und „Eindeutigkeit der Aufgabe“ korrelieren mit allen Fragen hochsignifikant positiv. Je besser das Verhältnis zum Arbeitgeber ist, desto mehr Freude hat der Software-Entwickler bei seiner Tätigkeit. Oder umgekehrt, je mehr Spass an der Arbeit der Programmierer hat, desto positiver sieht er seinen Arbeitgeber. Interessant ist der stark signifikante Zusammenhang der Professionalität des Arbeitgebers mit der Konzentration des Software-Entwicklers. Offensichtlich wirkt sich eine professionelle Vorgehensweise des Arbeitgebers positiv auf die Konzentration der Programmierer aus.

Ein eher überraschendes Bild ergibt sich bei der Korrelation der Flow-Komponenten mit den Fragen, die zur Arbeit im Software-Projekt gestellt wurden (Fragen 38 bis 42, siehe Tabelle 8.17). Alle Flow-Faktoren korrelieren signifikant positiv mit dem Druck, der von Abgabeterminen herrührt. Der bedeutendste Zusammenhang besteht zwischen den Komponenten „Flow/Spass“ und „Zeitempfinden/Immersion“. Dies lässt demnach die Schlussfolgerung zu, dass die Software-Entwickler umso mehr Spass an der Arbeit haben und in ihre Tätigkeit versinken, je mehr sie Abgabetermine verspüren. Dies widerspricht den Erkenntnissen von Amabile u. a. (1976), welche in ihrer empirischen Studie festgestellt haben, dass Abgabetermine die intrinsische Motivation der betroffenen Personen beeinträchtigen. Offensichtlich wirkt sich im Kontext von Software-Projekten der Abgabedruck nicht negativ auf den Spass aus. Im Gegenteil scheint ein Abgabetermin als Herausforderung wahrgenommen zu werden und führt dazu, dass sich die Programmierer auf ihre Tätigkeit konzentrieren, sich in die Arbeit hineinknien und auf diese Weise Flow erleben und Spass an der Tätigkeit haben.

Den Erwartungen entsprechend korreliert der Umstand, dass eine Projekt-Vision spürbar ist, hochsignifikant mit den Faktoren „Eindeutigkeit der Aufgabe“,

Tabelle 8.15: Flow-Empfinden und Engagement am Arbeitsplatz

Flow-Komponente	Korrelations-koeffizient	Signifikanz-niveau
29: Am Wochenanfang freue ich mich auf die bevorstehende Arbeit.		
Flow/Spass	0.509***	0.000
Konzentration	0.323***	0.000
Zeitempfinden/Immersion	0.322***	0.000
Eindeutigkeit der Aufgabe	0.224**	0.016
Herausforderung	0.169*	0.071
30: Für meine Arbeit Überstunden zu machen macht mir nichts aus.		
Flow/Spass	0.384***	0.000
Konzentration	0.233**	0.013
Zeitempfinden/Immersion	0.241***	0.010
Eindeutigkeit der Aufgabe	0.246***	0.009
Herausforderung	0.201**	0.033
31: Meine zukünftige Karriere plane ich im IT-Bereich.		
Flow/Spass	0.173*	0.071
Herausforderung	0.243**	0.011
Gesamtindex Engagement		
Flow/Spass	0.491***	0.000
Konzentration	0.292***	0.002
Zeitempfinden/Immersion	0.267***	0.005
Eindeutigkeit der Aufgabe	0.272***	0.004
Herausforderung	0.299***	0.002

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.16: Flow-Empfinden und Verhältnis zum Arbeitgeber

Flow-Komponente	Korrelations-koeffizient	Signifikanz-niveau
32: Ich bin stolz darauf, bei der Firma zu arbeiten.		
Flow/Spass	0.295***	0.002
Zeitempfinden/Immersion	0.196**	0.040
Eindeutigkeit der Aufgabe	0.305***	0.001
33: Ich kann mich bei der Firma persönlich und beruflich weiterentwickeln.		
Flow/Spass	0.351***	0.000
Zeitempfinden/Immersion	0.159*	0.093
Eindeutigkeit der Aufgabe	0.276***	0.003
34: Die Firma ist richtig aufgestellt und verfolgt die richtigen Ziele.		
Flow/Spass	0.261***	0.005
Eindeutigkeit der Aufgabe	0.251***	0.008
35: Bei der Firma werden das Rollenmodell und die Prozesse professionell und erfolgreich umgesetzt.		
Flow/Spass	0.341***	0.000
Konzentration	0.237**	0.013
Zeitempfinden/Immersion	0.204**	0.033
Eindeutigkeit der Aufgabe	0.318***	0.001
Gesamtindex Verhältnis zum Arbeitgeber		
Flow/Spass	0.346***	0.000
Zeitempfinden/Immersion	0.186*	0.056
Eindeutigkeit der Aufgabe	0.352***	0.000

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.17: Flow-Empfinden und Projektarbeit

Flow-Komponente	Korrelations-koeffizient	Signifikanz-niveau
38: Wie häufig sind Abgabetermine spürbar?		
Flow/Spass	0.255***	0.006
Konzentration	0.226**	0.016
Zeitempfinden/Immersion	0.269***	0.004
Eindeutigkeit der Aufgabe	0.174*	0.065
Herausforderung	0.177*	0.061
39: Wie oft sind die Software-Projekte von einer klaren Projektvision getragen?		
Flow/Spass	0.398***	0.000
Konzentration	0.324***	0.001
Zeitempfinden/Immersion	0.162*	0.092
Eindeutigkeit der Aufgabe	0.387***	0.000
40: Wie häufig ist die formale Autorität des Projektmanagers spürbar?		
Flow/Spass	0.179*	0.059
Konzentration	0.185*	0.050
41: Wie wichtig ist eine Vision hinter einem Software-Projekt für ihre Arbeit als Software-Entwickler?		
Flow/Spass	0.183*	0.052
Eindeutigkeit der Aufgabe	-0.186**	0.048

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

„Flow/Spass“ und „Konzentration“. Es leuchtet ein, dass eine nachvollziehbare Projektvision direkte Auswirkungen auf die Eindeutigkeit der Aufgabe hat. Mit Hilfe einer Projektvision können Unsicherheiten über die nächsten Schritte im Projekt-Alltag geklärt werden, sie beflügelt die Arbeiten im Projekt und hilft den Entwicklern, sich auf die Arbeit zu konzentrieren. Interessant ist, dass Software-Entwickler, die einen grösseren Anspruch an eine Projektvision haben, stark signifikant mehr Spass bei ihrer Tätigkeit empfinden. Dieser Anspruch korreliert dagegen negativ mit der Flow-Komponente „Eindeutigkeit der Aufgabe“. Allerdings ist dieser Zusammenhang nur schwach signifikant ($\alpha = 10\%$). Nicht unbedingt den Erwartungen entsprechend sind auch die positiven Korrelationen der im Projekt verspürten formalen Autorität mit „Flow/Spass“ und „Eindeutigkeit der Aufgabe“. Allerdings sind auch diese Korrelationen nur schwach signifikant.

Gibt es einen Zusammenhang zwischen Flow-Empfinden und Programmier-Erfahrung? Mit Hilfe der Angabe der Probanden, in welchem Jahr sie angefangen haben zu programmieren, konnte ich die Anzahl der Jahre an Programmier-

Tabelle 8.18: Flow-Empfinden und Erfahrung

Flow-Komponente	Korrelations-koeffizient	Signifikanz-niveau
Erfahrung		
Flow/Spass	0.188**	0.045
Konzentration	0.190**	0.043
Zeitempfinden/Immersion	0.170*	0.070

Bemerkung: ** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Erfahrung berechnen. Dieser Wert ergibt tatsächlich einige signifikante Korrelationen mit den Flow-Faktoren (siehe Tabelle 8.18). Erfahrene Software-Entwickler können sich demnach besser konzentrieren und erleben mehr Spass an ihrer Tätigkeit. Unter der Annahme, dass die Herausforderungen an professionelle Software-Entwickler gross ist, kann dies als Bestätigung der Hypothese verstanden werden, dass als Voraussetzung für Flow-Empfinden die Fähigkeiten den Anforderungen entsprechen müssen (siehe z.B. Csikszentmihalyi, 1975).

Diese Auswertungen lassen das Fazit zu, dass ein gutes Arbeitsumfeld positive Folgen auf die Freude des Software-Entwicklers an der Arbeit hat. Dies wiederum wirkt sich förderlich auf dessen Engagement aus. Zum positiven Arbeitsumfeld trägt neben dem guten Verhältnis zum Arbeitgeber eine nachvollziehbare Projektvision bei, während der Druck von Abgabeterminen überraschenderweise keine negativen Konsequenzen hat. Weiter hat auch die Erfahrung des Software-Entwicklers einen positiven Effekt auf dessen Spass an der Arbeit.

8.3.2 Weitere Korrelationen

Ein interessantes Bild wird aus der Tabelle 8.19 mit den Korrelationskoeffizienten aus Projektarbeit und Engagement ersichtlich. Die Tabelle zeigt einen stark signifikanten Zusammenhang zwischen der Einsatzbereitschaft und der Spürbarkeit von Projektvisionen (Ist-Zustand). Etwas weniger auffällig ist der Zusammenhang der Einsatzbereitschaft mit der Wichtigkeit von Projektvisionen (Soll-Zustand). Besonders ausgeprägt ist die Verknüpfung einer nachvollziehbaren Projektvision mit der Freude über die bevorstehende Arbeit. Auch die Bereitschaft, Überstunden zu machen, korreliert mit dem Soll-Zustand von Projektvisionen. Hingegen spielen Projektvisionen keine Rolle bei der Frage nach der zukünftigen Karriere im IT-Bereich. Für diese Frage entscheidend ist die verspürte formale Autorität im Software-Projekt sowie die fachliche Kompetenz der Vorgesetzten. Nicht überraschend wirkt sich der Umstand, dass formale Autorität spürbar ist, stark signifikant negativ auf die Vorstellungen über die weitere berufliche Entfaltung im IT-Bereich aus. Einen positiven Effekt auf die Karrierevorstellung hat hingegen die fachliche Kompetenz des Projektmanagers.

Tabelle 8.19: Projektarbeit und Engagement

Flow-Komponente	Korrelations-koeffizient	Signifikanz-niveau
29: Am Wochenanfang freue ich mich auf die bevorstehende Arbeit.		
Projektvisionen IST	0.306***	0.001
30: Für meine Arbeit Überstunden zu machen macht mir nichts aus.		
Projektvisionen IST	0.167*	0.084
31: Meine zukünftige Karriere plane ich im IT-Bereich.		
formale Autorität	-0.217**	0.024
fachliche Kompetenz	0.178*	0.064
Gesamtindex Engagement		
Projektvisionen IST	0.219**	0.025
Projektvisionen SOLL	0.176*	0.069

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Eine herausragende Bedeutung hat die Spürbarkeit einer Projektvision auf das Verhältnis zum Arbeitgeber. Wie aus der Tabelle 8.20 hervorgeht, besteht bei allen vier Aspekten, die bezüglich des Verhältnisses zum Arbeitgeber erhoben worden sind, ein hochsignifikant positiver Zusammenhang mit dem Umstand, dass die Entwicklungsarbeit durch eine Projektvision gestützt ist. Am deutlichsten wirkt dieser Zusammenhang bei der Einschätzung der Professionalität des Arbeitgebers. Interessant ist, dass mit dieser Frage auch die Wahrnehmung von formaler Autorität, wenn auch nur schwach signifikant, verknüpft ist. Möglicherweise braucht es für die Umsetzung einer professionellen Vorgehensweise ein zupackendes Management, was in Form von formaler Autorität positiv vermerkt wird. Bemerkenswert ist auch, dass der Druck von Abgabeterminen dem positiven Bild der Firma, bei welcher der Software-Entwickler arbeitet, keinen Abbruch tut: Der Stolz auf die Firma korreliert stark positiv mit der Häufigkeit von Abgabeterminen.

Ein erwarteter Zusammenhang ergibt sich bei der Korrelation der Fragen bezüglich des Verhältnisses zum Arbeitgeber mit dem Engagement am Arbeitsplatz (siehe Tabelle 8.21). Je besser das Verhältnis zum Arbeitgeber eingeschätzt wird, umso grösser ist die Einsatzbereitschaft. Die wichtigste Rolle spielt dabei der Umstand, dass sich der Software-Entwickler bei der Firma beruflich und persönlich weiterentwickeln kann. Bemerkenswert ist hier, dass die Frage, ob die zukünftige Karriere weiterhin im IT-Bereich stattfindet (Item 31), ganz unabhängig von den Erfahrungen beim aktuellen Arbeitgeber beantwortet wird.

In der Tabelle 8.22 sind die signifikanten Korrelationen von Projektsituation und Erfahrung aufgeführt. Die Tabelle zeigt einen hochsignifikanten Zusammen-

Tabelle 8.20: Projektarbeit und Verhältnis zum Arbeitgeber

Flow-Komponente	Korrelations-koeffizient	Signifikanz-niveau
32: Ich bin stolz darauf, bei der Firma zu arbeiten.		
Abgabetermine	0.191**	0.045
Projektvisionen IST	0.412***	0.000
Projektvisionen SOLL	0.161*	0.093
33: Ich kann mich bei der Firma persönlich und beruflich weiterentwickeln.		
Projektvisionen IST	0.465***	0.000
34: Die Firma ist richtig aufgestellt und verfolgt die richtigen Ziele.		
Projektvisionen IST	0.430***	0.000
35: Bei der Firma werden das Rollenmodell und die Prozesse professionell und erfolgreich umgesetzt.		
Projektvisionen IST	0.542***	0.000
formale Autorität	0.182*	0.059
Gesamtindex Verhältnis zum Arbeitgeber		
Projektvisionen IST	0.555***	0.000

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.21: Verhältnis zum Arbeitgeber und Engagement

Flow-Komponente	Korrelations-koeffizient	Signifikanz-niveau
29: Am Wochenanfang freue ich mich auf die bevorstehende Arbeit.		
32: Stolz darauf, bei der Firma zu arbeiten.	0.355^{***}	0.000
33: Persönliche und berufliche Entfaltung.	0.371^{***}	0.000
34: Die Firma ist aufgestellt.	0.250^{***}	0.008
35: Die Firma professionell.	0.189^{**}	0.048
Gesamtindex: Verhältnis zum Arbeitgeber	0.336^{***}	0.000
30: Für meine Arbeit Überstunden zu machen macht mir nichts aus.		
32: Stolz darauf, bei der Firma zu arbeiten.	0.345^{***}	0.000
33: Persönliche und berufliche Entfaltung.	0.408^{***}	0.000
34: Die Firma ist aufgestellt.	0.267^{***}	0.005
35: Die Firma professionell.	0.295^{***}	0.002
Gesamtindex Verhältnis zum Arbeitgeber	0.371^{***}	0.000
Gesamtindex Engagement		
32: Stolz darauf, bei der Firma zu arbeiten.	0.283^{***}	0.003
33: Persönliche und berufliche Entfaltung.	0.385^{***}	0.000
35: Die Firma professionell.	0.194^{**}	0.045
Gesamtindex Verhältnis zum Arbeitgeber	0.301^{***}	0.002

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.22: Projektsituation und Erfahrung

Flow-Komponente	Korrelations-koeffizient	Signifikanz-niveau
Erfahrung		
Abgabetermine	0.242^{***}	0.010
Projektvisionen SOLL	0.172^{*}	0.069

Bemerkung: *** Korrelation auf 1% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

hang zwischen Programmier-Erfahrung und der Wahrnehmung von Abgabeterminen. Je erfahrener der Software-Entwickler ist, desto häufiger nimmt er Abgabetermine wahr. Möglicherweise werden Programmierer intensiver in Software-Projekten eingesetzt, je mehr Erfahrung sie haben. Weiter lässt sich aus der Tabelle ableiten, dass ein Software-Entwickler mit zusätzlicher Erfahrung eine Projektvision immer mehr zu schätzen weiss. Dieser Zusammenhang ist allerdings nur schwach signifikant.

8.4 Typisierung der kommerziellen Software-Entwickler

8.4.1 Clusteranalysen

Die Fragebogen-Items zum Engagement (Fragen 29 - 31), zum Verhältnis zum Arbeitgeber (Fragen 32 - 35), zur Projekt-Situation (Fragen 38 - 42) und zur Rolle im Software-Projekt (Fragen 43 - 45) können im Hinblick auf eine Typisierung der Software-Entwickler mit Hilfe von Clusteranalysen untersucht werden.

Engagement am Arbeitsplatz

Eine partitionierende Clusteranalyse (K-Means-Methode) der Fragen zum Engagement ergibt die besten Resultate bezüglich Ausgewogenheit der Cluster, Stabilität und Konsistenz bei einer Lösung mit einer aus zwei Clustern bestehenden Lösung.

Die Clusteranalyse konvergiert in vier Iterationen. Die Interpretation der Cluster auf Grund der Analyse der Clusterzentren liegt auf der Hand (siehe Tabelle 8.23): Der erste Cluster hat in jedem Item einen höheren Wert als der zweite Cluster. Der erste Cluster wird demnach durch die *engagierten* Software-Entwickler gebildet, während der zweite Cluster die *distanzierten* Entwickler umfasst. Im Sample der kommerziellen Software-Entwickler war die Menge der engagierten Programmierer mit 65% fast doppelt so gross wie der Anteil der distanzierten Entwickler.

Verhältnis zum Arbeitgeber

Die Clusteranalyse mit den Frageitems 32 - 35 konvergiert am besten mit vier Clustern. Bei einer solcher Wahl ist die Konvergenz nach vier Iterationen erreicht.

Die Analyse der Clusterzentren nach der Iteration legt folgende Interpretation der Cluster nahe (siehe Tabelle 8.24): Der erste Cluster weist überall die höchsten Werte auf. Die Probanden in diesem Cluster sind *stolz* auf ihren Arbeitgeber. Dieser Cluster umfasst 28% der Entwickler im Sample. Der zweite Cluster hat in jedem Item einen Punkt weniger verglichen mit dem ersten Cluster. Die Programmierer in diesem Cluster können als *mässig stolz* auf ihren Arbeitgeber bezeichnet werden (Anteil 24% des Samples). Der dritte Cluster hat überall die schlechtesten Werte. Die Software-Entwickler in diesem Cluster zeichnen sich durch ein *distanziertes* Verhältnis zu ihrem Arbeitgeber aus. Zu dieser Gruppe gehören 18% der befragten Programmierer. Die Entwickler im letzten Cluster beantworteten die ersten beiden Fragen (bezüglich Stolz und persönlicher Entfaltung) mit hohen Werten, während sie sich bei der Beurteilung der Firma bezüglich Zielsetzung und Professionalität kritisch zeigten. Die Gruppe der *kritisch zufriedenen* Programmierer umfasst 30% des Samples.

Situation im Software-Projekt

Eine Clusteranalyse der Items zur Situation im Software-Projekt (38-42) konvergierte am gleichmässigsten mit vier Clustern (in 10 Iterationen). Mit einer Varianzanalyse kann gezeigt werden, dass die Frage 40 am besten diskriminiert.

Die Durchsicht der ermittelten Clusterzentren (siehe Tabelle 8.25) legt folgende Interpretation nahe: Die Software-Entwickler im ersten Cluster zeichnen sich dadurch aus, dass sie am wenigsten Ansprüche an die Projekt-Vision haben und, zusammen mit dem vierten Cluster, am wenigsten unter Abgabeterminen leiden. Diese Software-Entwickler können als *pragmatisch* charakterisiert werden. Die Entwickler im zweiten Cluster haben einen hohen Anspruch an die Projekt-Vision, stellen aber gleichzeitig fest, dass diese Erwartung häufig erfüllt wird (Item 41 bzw. 39). Weiter finden sie die fachliche Kompetenz der Projektmanager wichtig, verspüren aber nur in wenigen Fällen, dass der Projektleiter seine formale Autorität ausspielt (Item 42 bzw. 40). Auf Grund dieser Feststellungen können die Software-Entwickler in diesem Cluster als *zufrieden* bezeichnet werden. Im Gegensatz dazu steht der dritte Cluster. Die Antworten in diesem Cluster zeigen eine grosse Diskrepanz zwischen Soll- und Ist-Zustand: Die hohen Ansprüche an die Projekt-Vision werden nur in geringem Mass erfüllt und statt Fachkompetenz erleben diese Entwickler häufig die Projektleiter als formale Autoritäten. Gleichzeitig leiden diese Entwickler in hohem Mass unter Abgabeterminen. Dieser Cluster kann demnach als Gruppe der *unzufriedenen* Software-Entwickler interpretiert werden. Der vierte Cluster wird von den *visionär enttäuschten* Software-Entwicklern gebildet. Diese haben einen hohen Anspruch an die Projekt-Vision und sehen diesen Anspruch in sehr geringem Mass erfüllt. Hingegen sind sie zufrieden bezüglich der Projektleitung (sie nehmen wenig formale Autorität wahr) und bezüglich der Häufigkeit von Abgabeterminen.

Rolle im Software-Projekt

Mit den Fragen 43 - 45 kann die Rolle der Probanden im Software-Projekt bestimmt werden. Mit Hilfe der Clusteranalyse ist es einfach, jedem Datensatz eine Rolle zuzuweisen. Die Anzahl der Cluster ist in diesem Fall auf Grund der möglichen Rollen (Projektleiter, Software-Architekt, Entwickler) vorgegeben. Eine möglichst gleichmässige Aufteilung der Antworten auf die Cluster ist nicht notwendig.

Diese Clusteranalyse konvergiert in 6 Iterationen. Die Tabelle mit den Clusterzentren (siehe Tabelle 8.26) zeigt das gewünschte Bild: Bei Cluster 1 handelt es sich um Projektleiter, beim zweiten Cluster um Software-Architekten, während der dritten Cluster die gewöhnlichen Software-Entwickler umfasst. Diese Analyse zeigt, dass die Umfrage von 33 Projektleitern (31%), 11 Software-Architekten (10%) und 62 Entwicklern (59%) beantwortet wurde.

Tabelle 8.23: Clusteranalyse der Fragen zum Engagement

Item	**Cluster 1** (engagiert)	**Cluster 2** (distanziert)
29: Am Wochenanfang freue ich mich auf die bevorstehende Arbeit.	4	3
30: Für meine Arbeit Überstunden zu machen macht mir nichts aus.	5	3
31: Meine zukünftige Karriere plane ich im IT-Bereich.	5	4
Anteil des Clusters im Sample	65%	35%

Bemerkung: n = 109

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.24: Clusteranalyse der Fragen zum Verhältnis zum Arbeitgeber

Item	**Cluster 1** (stolz)	**Cluster 2** (mässig st.)	**Cluster 3** (dist.)	**Cluster 4** (krit. zufr.)
32: Ich bin stolz darauf, bei der Firma zu arbeiten.	5	4	2	4
33: Ich kann mich bei der Firma persönlich und beruflich weiterentwickeln.	5	4	2	4
34: Die Firma ist richtig aufgestellt und verfolgt die richtigen Ziele.	5	4	2	3
35: Bei der Firma werden das Rollenmodell und die Prozesse professionell und erfolgreich umgesetzt.	4	3	2	2
Anteil des Clusters im Sample	28%	24%	18%	30%

Bemerkung: n = 106

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.25: Clusteranalyse der Fragen zur Projektsituation

Frage	**Cluster 1** (pragm.)	**Cluster 2** (zufrieden)	**Cluster 3** (unzufr.)	**Cluster 4** (vis. ent.)
38: Wie häufig sind Abgabetermine spürbar?	4	5	6	4
39: Wie oft sind die Software-Projekte von einer klaren Projektvision getragen?	4	5	3	2
40: Wie häufig ist die formale Autorität des Projektmanagers spürbar?	4	3	5	2
41: Wie wichtig ist eine Vision hinter einem Software-Projekt für ihre Arbeit als Software-Entwickler?	4	5	5	5
42: Wie wichtig ist die fachliche Kompetenz des Projektmanagers für Ihre Arbeit in einem Software-Projekt?	5	5	5	5
Anteil des Clusters im Sample	31%	28%	20%	21%

Bemerkung: n = 114

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.26: Rolle im Software-Projekt

Frage	**Cluster 1** (P.-Leiter)	**Cluster 2** (S.-Arch.)	**Cluster 3** (S.-Entw.)
43: Wie viel Zeit an Ihrem Arbeitsplatz sind Sie als Projektleiter tätig?	4	1	1
44: Wie viel Zeit an Ihrem Arbeitsplatz sind Sie als Software-Architekt tätig?	3	8	2
45: Wie viel Zeit an Ihrem Arbeitsplatz sind Sie als Software-Entwickler tätig?	3	6	8
Clustergrösse	33	11	62
Anteil des Clusters im Sample	31%	10%	59%

Bemerkung: n = 106
1: 0% - 10%, 2: 11% - 20%, 3: 21% - 30%, 4: 31% - 40% etc.

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.27: Kreuztabelle: Engagement vs. Verhältnis zum Arbeitgeber

	Engagement		
Verhältnis	engagiert	distanziert	Total
stolz	26	8	34
	(17.8)	(16.2)	
mässig stolz	13	16	29
	(15.1)	(13.9)	
distanziert	6	14	20
	(10.4)	(9.6)	
kritisch zufrieden	14	16	30
	(15.7)	(14.3)	
Total	59	54	113

Bemerkung: In Klammern die erwartete Anzahl

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

8.4.2 Verifikation der Typisierungen

Die mit Hilfe der Clusteranalysen im vorherigen Abschnitt 8.4.1 gemachten Typisierungen der kommerziellen Software-Entwickler sind als solche ad-hoc und intuitiv. Mit zusätzlichen Untersuchungen will ich zeigen, dass die Interpretationen der verschiedenen Cluster durchaus vernünftig sind.

Wird das Engagement (Tabelle 8.23) mit dem Verhältnis zum Arbeitgeber (Tabelle 8.24) kombiniert, so bestätigt der Vergleich der tatsächlichen mit den erwarteten Häufigkeiten meine Interpretationen weitgehend.

Von einem Software-Entwickler, der ein gutes Verhältnis zum Arbeitgeber hat und Stolz zeigt, bei dieser Firma mitwirken zu können, erwarten wir ein grosses Engagement. Umgekehrt erwarten wir von einem Programmierer, der ein distanziertes Verhältnis zu seinem Arbeitgeber hat, eine ebenso distanzierte Arbeit. Die Kreuztabelle 8.27 bestätigt diese Annahmen. Die Abweichungen von den erwarteten Werten sind sowohl für die stolzen wie auch für die distanzierten Software-Entwickler markant, während mässig stolze wie auch kritisch zufriedene Programmierer bezüglich des Engagements nahe bei den erwarteten Werten liegen. Der χ^2-Test ist für die Tabelle hochsignifikant.

Analoge Annahmen können wir machen, wenn wir das Engagement mit der Arbeitssituation (Tabelle 8.25) kombinieren. Von zufriedenen Entwicklern erwarten wir ein grösseres Engagement, während sich pragmatische Entwickler wohl durch ein distanzierteres Engagement auszeichnen. Die Kreuztabelle 8.28 bestätigt diese Annahmen weitgehend. Die tatsächlichen Häufigkeiten der zufriedenen bzw. pragmatischen Entwickler weichen bezüglich Engagement markant von den erwarteten Häufigkeiten ab. Die Abweichungen sind statistisch hochsignifikant (Test von χ^2 mit $p < 1\%$). Interessant ist, dass sowohl die unzufriedenen wie auch die visionär enttäuschten Software-Entwickler bezüglich Engagement praktisch mit den Erwartungen übereinstimmen.

Tabelle 8.28: Kreuztabelle: Engagement vs. Projektsituation

	Engagement		
Projektsituation	engagiert	distanziert	Total
pragmatisch	12 (18.3)	23 (16.7)	35
zufrieden	24 (16.2)	7 (14.8)	31
unzufrieden	12 (12.0)	11 (11.0)	23
visionär enttäuscht	11 (12.5)	13 (11.5)	24
Total	59	54	113

Bemerkung: In Klammern die erwartete Anzahl

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Auch eine Kombination von Engagement und Projektrolle (Tabelle 8.26) kann als Bestätigung meiner Interpretationen verstanden werden. Projektleiter und Software-Architekten zeigen sich häufiger engagiert, während die normalen Software-Entwickler eher distanziert arbeiten (siehe Kreuztabelle 8.29). Allerdings sind diese Differenzen statistisch nicht signifikant.

Statistisch hochsignifikant sind die Abweichungen der tatsächlichen von den erwarteten Häufigkeiten in der Kreuztabelle, die sich aus der Kombination des Verhältnisses zum Arbeitgeber mit der Projekt-Situation ergibt (Tabelle 8.30 mit einem χ^2-Test von $p < 1\%$). Diese Kombination zeigt die pragmatischen Software-Entwickler mässig stolz auf ihre Arbeitgeber, während die zufriedenen Programmierer stolz auf ihren Arbeitsplatz sind und ein wenig distanziertes Verhältnis haben. Die Entwickler, die mit der Projektsituation unzufrieden sind, sind auch nur kritisch zufrieden mit dem Arbeitgeber, während die Entwickler, die eine Vision bei ihrer Projektarbeit vermissen, ein distanziertes Verhältnis gegenüber dem Arbeitgeber haben.

Bei den beiden übrigen möglichen Kombinationen (Projekt-Rolle vs. Verhältnis zum Arbeitgeber und Projekt-Rolle vs. Projekt-Situation) ergibt der χ^2-Test keine signifikanten Werte.

8.4.3 Zusammenhang mit Flow-Empfinden

Erleben die verschiedenen Typen von Software-Entwicklern ihre Tätigkeit unterschiedlich? Werden die Mittelwerte der Flow-Komponenten in Abhängigkeit der Entwickler-Typen untersucht, so können tatsächlich charakteristische Unterschiede festgestellt werden.

Bezüglich des Engagements zeigt sich das erwartete Verhalten. Engagierte Software-Entwickler zeigen in allen Flow-Komponenten bessere Werte (Abbildung 8.2). Die Unterschiede sind statistisch allesamt auf dem 1%-Nivau signifikant (Ta-

Tabelle 8.29: Kreuztabelle: Engagement vs. Projekt-Rolle

	Engagement		
Rolle	engagiert	distanziert	Total
Projektleiter	19 (16.8)	13 (15.2)	32
Software-Architekt	8 (5.8)	3 (5.2)	11
Entwickler	28 (32.5)	34 (29.5)	62
Total	55	50	105

Bemerkung: In Klammern die erwartete Anzahl

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 8.30: Kreuztabelle: Verhältnis zum Arbeitgeber vs. Projektsituation

	Verhältnis zum Arbeitgeber				
Projektsituation	stolz	mässig stolz	distanziert	krit. zufr.	Total
pragmatisch	7 (10.5)	15 (9.0)	4 (6.2)	9 (9.3)	35
zufrieden	17 (9.3)	6 (8.0)	1 (5.5)	7 (8.2)	31
unzufrieden	7 (6.9)	2 (5.9)	5 (4.1)	9 (6.1)	23
visionär enttäuscht	3 (7.2)	6 (6.2)	10 (4.2)	5 (6.4)	24
Total	34	29	20	30	113

Bemerkung: In Klammern die erwartete Anzahl

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

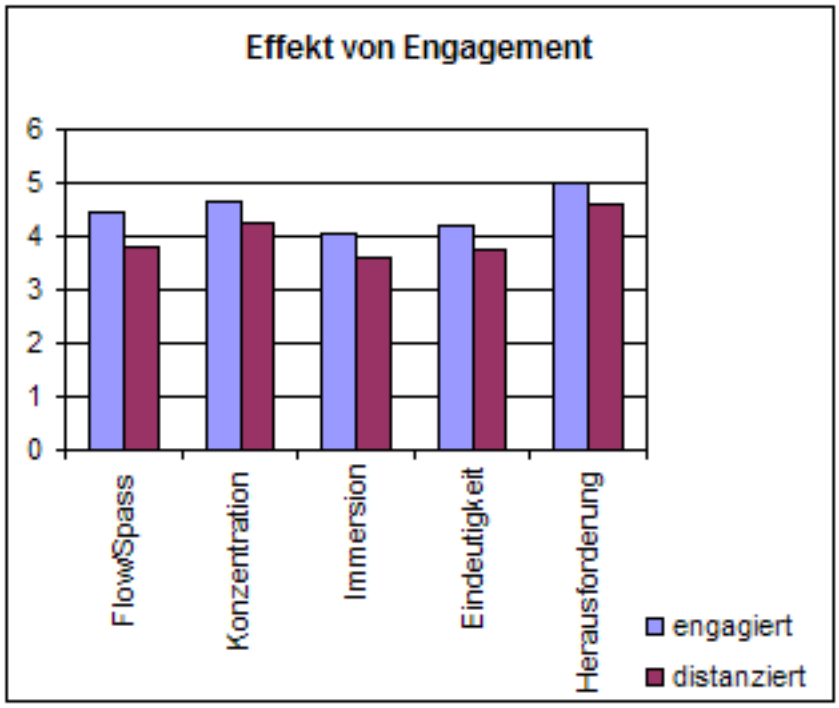


Abbildung 8.2: Flow-Empfinden in Abhängigkeit des Engagements

belle 8.31).

Die Interpretation der Entwickler-Typen auf Grund des Verhältnisses zum Arbeitgeber wird ebenfalls unterstützt, wenn wir das Flow-Empfinden dieser Typen untersuchen. Wir erwarten, dass Software-Entwickler, die stolz auf ihren Arbeitgeber sind und sich in ihrem Arbeitsumfeld wohl fühlen, mehr Spass haben an ihrer Tätigkeit und häufiger Flow erleben. Mit Ausnahme der Komponente „Herausforderung“, wo keine signifikanten Unterschiede zwischen den Typen bestehen, zeigen die auf ihren Arbeitgeber stolzen Software-Entwickler überall die höchsten Werte (Abbildung 8.3). Ein T-Test für die Gleichheit der Mittelwerte bestätigt, dass der Mittelwert der stolzen Entwickler signifkant über den Mittelwerten der anderen Typen liegt für die Komponenten „Flow/Spass“, „Konzentration“, „Immersion“ und „Eindeutigkeit der Aufgabe“ (Tabelle 8.32).

Werden die Mittelwerte der verschiedenen Flow-Komponenten in Abhängigkeit der Projekt-Rolle dargestellt, so wird ersichtlich, dass tendenziell die Software-Architekten Flow intensiver erleben (siehe Abbildung 8.4), wieder mit Ausnahme der Komponente „Herausforderung“. Herausforderung wird am meisten von den Projektleitern verspürt. Dieses Ergebnis ist intuitiv nachvollziehbar. Statistisch signifikant ist allerdings einzig der Unterschied in der Komponente „Herausforderung“, und in diesem Fall nur die Differenz zwischen dem Projektleiter und dem Software-Entwickler ($\alpha = 5\%$).

Ein auf den ersten Blick uneinheitliches Bild ergibt sich, wenn wir das Flow-Erleben der Typen betrachten, die sich auf Grund der Projektsituation ergeben.

Tabelle 8.31: Flow-Empfinden in Abhängigkeit des Engagements

T-Test für die Mittelwertgleichheit			
Flow-Faktor	**Differenz**	**Signifikanz**	**T-Wert**
Flow/Spass	0.642	0.00^{***}	4.829
Konzentration	0.393	0.00^{***}	2.963
Zeitempfinden/Immersion	0.453	0.01^{***}	2.768
Eindeutigkeit der Aufgabe	0.415	0.00^{***}	2.927
Herausforderung	0.395	0.00^{***}	3.054

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

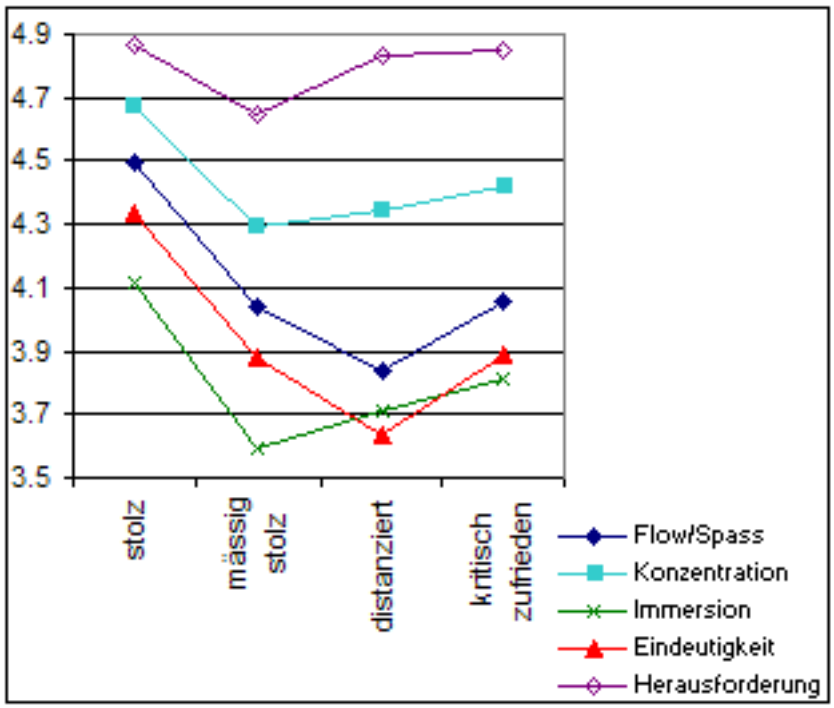


Abbildung 8.3: Flow-Empfinden und Verhältnis zum Arbeitgeber

Tabelle 8.32: Flow-Empfinden und Verhältnis zum Arbeitgeber

T-Test für die Mittelwertgleichheit des Typs „stolz“		
Flow-Faktor	**Signifikanz**	**T-Wert**
Flow/Spass	0.00***	-3.260
Konzentration	0.03**	-2.181
Zeitempfinden/Immersion	0.05*	-1.994
Eindeutigkeit der Aufgabe	0.00***	-3.310
Herausforderung	0.53	-0.627

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

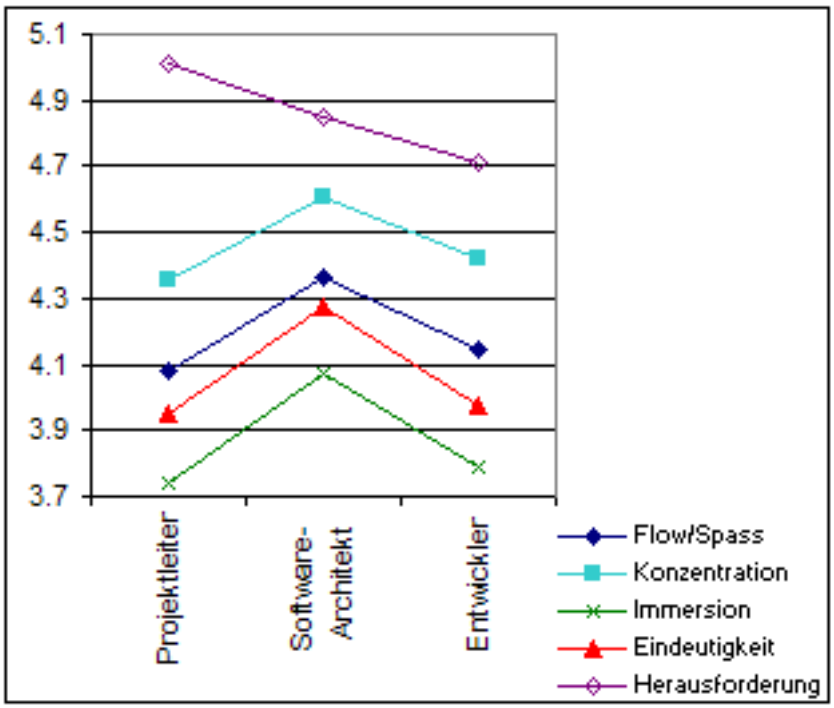


Abbildung 8.4: Flow-Empfinden in Abhängigkeit der Projekt-Rolle

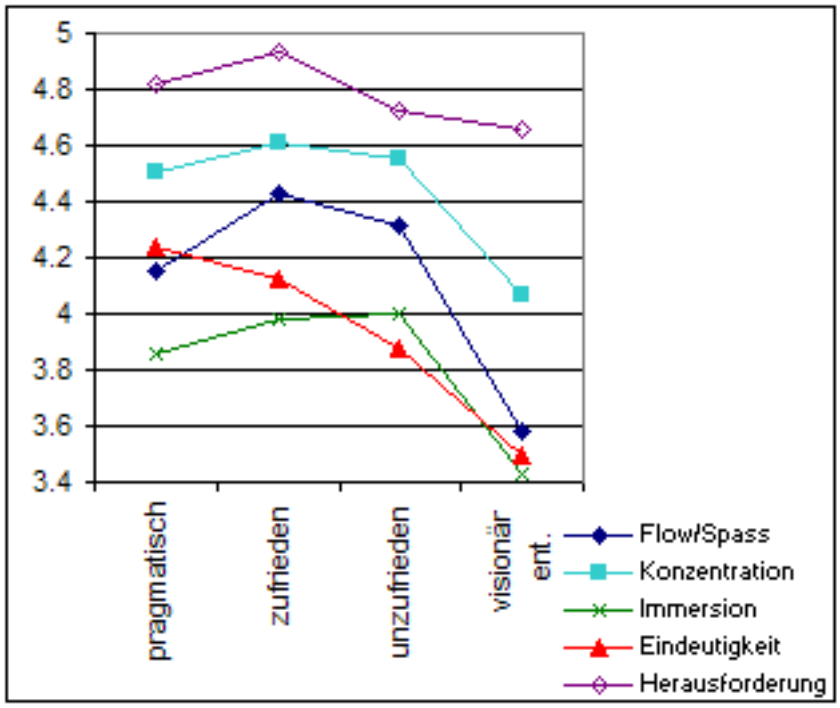


Abbildung 8.5: Flow-Empfinden in Abhängigkeit der Projektsituation

Wieder erwarten wir, dass die zufriedenen Software-Entwickler die höchsten Werte zeigen. Diese Erwartung wird enttäuscht: Die besten Werte zeigen die zufriedenen Programmierer nur für die Komponenten „Flow/Spass“, „Konzentration“ und „Herausforderung“ (siehe Abbildung 8.5). In allen Fällen unterscheidet sich der Mittelwert der zufriedenen Programmierer nicht signifikant von den anderen Typen.

Hingegen fällt auf, dass der visionär enttäuschte Entwickler-Typ für jede Komponente den schlechtesten Wert aufweist. Der T-Test für die Mittelwertgleichheit bestätigt, dass (bis auf die Komponente „Herausforderung“) der Unterschied signifikant ist (minimal auf 5%-Niveau, siehe Tabelle 8.33). Wenn wir uns erinnern, dass der „visionäre“ Entwickler-Typ charakterisiert ist durch eine grosse Diskrepanz zwischen wahrgenommener und erwünschter Projekt-Vision, so weist dieses Resultat darauf hin, dass eine fehlende oder ungenügende Projekt-Vision dafür verantwortlich sein kann, dass Software-Entwickler den Spass an der Tätigkeit verlieren.

8.5 Folgerungen

Die Auswertungen in diesem Kapitel zeigen, dass die Fragen nach dem Engagement, dem Verhältnis zum Arbeitgeber und der Projektsituation mit einer Clusteranalyse untersucht werden können und dass auf diese Weise ein Typisierung der Software-Entwickler erreicht werden kann, die nachvollziehbar ist und als

Tabelle 8.33: Flow-Empfinden in Abhängigkeit der Projektsituation

T-Test für die Mittelwertgleichheit des Typs „visionär enttäuscht“		
Flow-Faktor	**Signifikanz**	**T-Wert**
Flow/Spass	0.00^{***}	-4.292
Konzentration	0.01^{**}	-2.614
Zeitempfinden/Immersion	0.01^{**}	-2.551
Eindeutigkeit der Aufgabe	0.00^{***}	-3.580
Herausforderung	0.26	-1.124

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Basis für weitere Untersuchungen dienen kann. Im Grossen und Ganzen kann unterschieden werden in Software-Entwickler, die engagiert arbeiten, stolz sind auf die Firma, bei welcher sie angestellt sind, und die mit ihrer Tätigkeit zufrieden sind auf der einen Seite. Auf der anderen Seite arbeiten die Programmierer distanziert, ihr Verhältnis zum Arbeitgeber ist kritisch und sie sind mit der Arbeitssituation unzufrieden oder enttäuscht. Konsistent mit dieser Interpretation weisen die unterschiedlichen Typen signifikant verschiedene Werte bezüglich der Flow-Komponenten auf. Die zufriedenen und stolzen Programmierer haben in allen Belangen mehr Spass an ihrer Tätigkeit als die distanzierten und enttäuschten Software-Entwickler.

Während ich in der Open-Source-Umfrage die Produktivität der Software-Entwickler mittels der Anzahl geschriebener Patches bzw. Module zu messen versuchte, erkundigte ich mich in der kommerziellen Umfrage nach der Anzahl Checkins in den letzten Arbeitstagen. Auch in diesem Fall konnte ich keinerlei Korrelationen mit dem Empfinden von Spass nachweisen. Dieses Ergebnis kann auf zwei unterschiedliche Arten interpretiert werden. Entweder korrelieren Spass und Produktivität, aber meine Operationalisierung von Produktivität mittels der Anzahl produzierter Module oder der Häufigkeit von Checkins erfüllt ihren Zweck, Produktivität zu messen, nicht. Die zweite Interpretation wäre, dass meine Operationalisierung der Produktivität von Programmierern korrekt ist und das Resultat der statistischen Auswertung genau den Sachverhalt wiedergibt, dass Spass und Produktivität nicht korrelieren. Dies würde dann bedeuten, dass ein Programmierer Spass an der Tätigkeit hat, ohne dabei aber produktiv zu arbeiten und umgekehrt.

Mir erscheint die erste Interpretation überzeugender. Eine Person erlebt Flow, wenn ihre Fähigkeiten in Übereinstimmung mit den Herausforderungen sind (siehe Abschnitt 4.1). Die Herausforderung besteht aus irgendeiner Art von Problem, das zu lösen ist. Bei einem Computerspiel wie Tetris besteht das Problem beispielsweise darin, auf dem Bildschirm angezeigten geometrischen Formen, die in immer grösserer Zahl und mit grösserer Geschwindigkeit fallen, vor dem Bodenkontakt in eine lückenlose Linie zu bringen. Beim Software-Entwickeln besteht das Problem beispielsweise aus einem Fehler in der Applikation oder aus einer noch nicht in der Applikation enthaltenen Funktionalität. Ein „unproduktiver“ Tetris-Spieler wird nur kurze Spielsequenzen spielen können, da der Computer auf Grund von

mangelnden feinmotorischen Fähigkeiten und ungenügender Reaktionsschnelligkeit des Spielers schon bald *game over* melden wird. Ein solcher Spieler wird keinen Spass beim Spielen empfinden können, sondern das Spiel nach kurzer Zeit frustriert weglegen. Analog wird ein unproduktiver Programmierer, dem es nicht gelingt, einen Software-Fehler zu finden, oder der an der Aufgabe scheitert, eine neue Funktionalität zu implementieren, die Arbeitsepisode als frustrierend empfinden.

Nicht jede Arbeit kann als Problemlösung interpretiert werden. Für das Entwickeln von Software spielt aber das Lösen von Problemen eine zentrale Rolle. Auf Grund dieser Überlegungen bin ich der Ansicht, dass ein Programmierer unmöglich Spass am Programmieren hat, wenn er gleichzeitig daran scheitert, die Aufgabe zu lösen. Ein Programmierer, der in seiner Arbeit Flow erlebt, ist deshalb notwendigerweise auch produktiv. Mit dieser Überlegung könnte aber der umgekehrte Fall möglich sein: Ein Software-Entwickler ist produktiv und löst die gestellten Aufgaben, ohne dass er dabei aber Spass empfindet. Dies könnte beispielsweise dann passieren, wenn der Programmierer auf Grund seiner Fähigkeiten und seiner Erfahrung das Problem mit Leichtigkeit löst, die Arbeit also nicht als Herausforderung erlebt, sonder als langweilig empfindet. Ich bin allerdings der Ansicht, dass eine solche Produktivität nur kurzzeitig aufrechterhalten werden könnte. Die im Falle von Unterforderung zwangsläufig eintretenden Langeweile zerstört die Konzentration der Programmierers, was unvermeidlich auch negative Folgen für die Produktivität dieses Entwicklers haben wird.

Auf Grund dieser Betrachtungen bin ich deshalb der Überzeugung, dass meine in der Untersuchung verwendete Operationalisierung von Produktivität (Anzahl erzeugte Module bzw. Anzahl Checkins) kein gutes Mass ist, diese Variable zu messen. Wird Produktivität als Problemlösungskapazität verstanden, so wird auch nachvollziehbar, weshalb meine Operationalisierung wenig geeignet ist für diesen Zweck. Zur Behebung eines Fehlers muss möglicherweise nur eine Zeile im Code geändert werden. Trotzdem kann es sich dabei um einen äusserst subtilen und schwierig zu findenden Fehler handeln. Neue Funktionalität muss je nach verwendeter Technologie und je nach Art des Problems in vielen verschiedenen Modulen implementiert werden oder kann mit Hilfe eines knappen Stücks Code umgesetzt werden. Wenig erfahrene Programmierer brauchen möglicherweise eine ganze Reihe von Checkins, bis ein Modul fehlerfrei läuft, während fortgeschrittene Software-Entwickler ein konkretes Problem auf Anhieb fehlerfrei realisieren. Diese Erläuterungen zeigen, dass die erhobenen Zahlen wenig aussagekräftig sind. Je nach den Umständen und der Art des Problems reflektiere hohe Zahlen eine grosse Produktivität oder auch gerade das Gegenteil.

Es stellt sich demnach die Frage, wie ein gültiges Mass für Produktivität aussehen könnte. Mit den aus dieser Untersuchung gewonnenen Erfahrungen erscheint es mir naheliegend, Produktivität als Zusammenspiel von individuellen Fähigkeiten, verwendeten Arbeitsmitteln und Komplexitätsgrad der zu lösenden Aufgaben zu modellieren. Bei gleichbleibenden Fähigkeiten und Arbeitsmitteln weist eine Person eine grössere Produktivität auf, wenn die pro Zeiteinheit gelöste Anzahl Probleme von grossem Schwierigkeitsgrad sind als wenn es sich um einfache Probleme handelt. Bei gegebenen Fähigkeiten und fixiertem Schwierigkeitsgrad der Aufgaben wird die Produktivität der Person grösser sein, wenn sie über leistungsfähige

Werkzeuge verfügt, als wenn sie mit veralteten Mitteln arbeiten muss. Bei gegebenen Arbeitsmitteln und fixiertem Schwierigkeitsgrad der Aufgaben wird die Produktivität einer Person mit wachsenden Fähigkeiten zunehmen.

Gemäss Asendorpf (1996, S. 137) sind Fähigkeiten „Persönlichkeitsmerkmale, die Leistungen ermöglichen". Sie bestehen einerseits aus sozialen und genetischen Komponenten, können aber auch mit Ausbildung und Erfahrung erworben werden. Unter den eingesetzten Arbeitsmitteln verstehe ich im Falle von Software-Entwicklern eine Kombination von Hardware-Infrastruktur (Leistungsfähigkeit der Prozessoren der Computer, Transferraten beim Austausch von Daten über das Internet etc.) und den für den Software-Entwicklungsprozess gebrauchten Programmen (z.B. Code-Editor bzw. integrierte Entwicklungsumgebung, Versionenverwaltung etc.). Um die Produktivität eines Software-Entwicklers vorauszusagen, würde ich beispielsweise dessen Erfahrung und Ausbildung sowie die von ihm verwendeten (oder ihm zur Verfügung gestellten) Arbeitswerkzeuge ermitteln. Die zu bewältigenden Aufgaben bzw. deren Schwierigkeitsgrad sind exogen gegeben.

Welcher Zusammenhang besteht nun zwischen Produktivität gemäss dieser Auslegeordnung und Spass? Meiner Meinung nach ist Produktivität über den Faktor „Leistungsbereitschaft" mit Spass verknüpft. Leistungsbereitschaft taucht aber in meiner Liste der Faktoren, welche die Produktivität einer Person konstituieren, nicht auf. Dies zu gutem Recht: Meines Erachtens ergibt sich Leistungsbereitschaft aus einer Übereinstimmung zwischen den Fähigkeiten, d.h. der Leistungsfähigkeit einer Person mit dem Schwierigkeitsgrad der an sie gestellten Aufgaben.[3] Eine Person mit unzureichenden Fähigkeiten ist überfordert und kann die geforderte Leistung nicht erbringen. Eine unterforderte Person ist umgekehrt gelangweilt und wird aus diesem Grund ihr Leistungspotential nicht ausreizen. Mit dem Element „Leistungsbereitschaft" wird also der Faktor „Fähigkeit" in eine Beziehung zum Faktor „Komplexitätsgrad der Aufgabe" gesetzt. Damit eine Person produktiv ist, muss sie nicht nur über die geeigneten Arbeitsmittel und Fähigkeiten verfügen, darüber hinaus müssen die Fähigkeiten mit dem Schwierigkeitsgrad der Aufgaben korrespondieren.

Wie aber die Flow-Forschung gezeigt hat, setzt das Empfinden von Flow seinerseits eine Übereinstimmung von Fähigkeiten und Herausforderung voraus (siehe Kapitel 4.1). Wenn nun aber Leistungsbereitschaft gleichermassen wie das Erleben von Flow eine optimale Übereinstimmung zwischen Fähigkeiten und Herausforderungen anzeigt und diese Übereinstimmung, die geeigneten Arbeitsmittel vorausgesetzt, die Produktivität einer Person definiert, so bedeutet dies nichts anderes, als dass der während der Arbeit empfundene Spass als Stellvertreter für die Produktivität der Person dienen kann. Damit wird aber die ursprüngliche Frage, ob sich Spass an der Arbeit positiv auf die Produktivität auswirkt, und die abgeleitete Frage, wie Produktivität gemessen werden kann, um diese Hypothese zu testen, hinfällig. Spass selbst ist das gesuchte Mass für Produktivität.

Sollte diese Ableitung, dass Spass ein Mass für die Produktivität eines Software-Entwicklers ist, zutreffen, so erhält der Befund, dass zufriedene und stolze Programmierer mehr Spass haben, eine zusätzliche Bedeutung. Investitionen des Ar-

[3] Möglicherweise gibt es auch noch kulturelle und gesundheitliche Faktoren, welche die Leistungsbereitschaft eines Individuums beeinflussen.

beitgebers in die Zufriedenheit und den Stolz der Programmierer müssten sich dann in Form von erhöhter Produktivität der Entwickler auszahlen. Folgende Empfehlungen lassen sich demnach für Software-Firmen ableiten:

- Fördern Sie den Stolz in die Firma: Unterstützen Sie die Programmierer bei der beruflichen Weiterentwicklung, geben Sie darauf acht, dass die firmeninternen Prozesse professionell und effizient umgesetzt werden und sorgen Sie für eine aufgestellte Stimmung am Arbeitsplatz.
- Liefern Sie Projekt-Visionen: Erklären Sie den Programmierern, welche Probleme mit der neuen Software gelöst werden können und welcher Mehrwert (für den Kunden, für die Gesellschaft) damit entsteht.
- Und vor allem: Verschaffen Sie sich ein klares Bild über die Fähigkeiten und das Potential Ihrer Programmierer und versuchen Sie, den Entwicklern mit ihrer Arbeit richtige Herausforderungen zu bieten.

Kapitel 9

Vergleich der Open-Source- und kommerziellen Entwickler

Nach diesen Vorarbeiten, der gründlichen Analyse der beiden Datensätze, kann ich nun daran gehen, diese Datensätze zu kombinieren und die Antworten der beiden Populationen zu vergleichen. Ziel dieser Analyse ist die Verifikation der Hypothese, dass das Entwickeln von Software mehr Spass macht, wenn diese Tätigkeit in einem Open-Source-Umfeld ausgeübt wird als wenn unter kommerziellen Bedingungen programmiert wird (Forschungsfrage 3).

Auch wenn das Design der Studie mit zwei analogen, in Teilen sogar identischen Umfragen, auf Vergleichbarkeit ausgelegt worden ist, muss bei den jeweiligen Gegenüberstellungen sorgfältig abgeklärt werden, ob nicht trotzdem systematische Unterschiede vorliegen, welche allfällig festgestellte Differenzen begründen könnten. Immerhin fand die Open-Source-Umfrage im Frühjahr 2004 statt, die kommerzielle Umfrage ein halbes Jahr später. Die Open-Source-Umfrage richtete sich an ein weltweites Publikum und wurde vorwiegend in englischer Sprache ausgefüllt, während die Teilnehmer der kommerziellen Umfrage deutschsprachige Software-Entwickler in Schweizer Firmen waren. Die Open-Source-Programmierer, die an der Umfrage teilnahmen, waren ausgesprochen selbstselektiert, während die kommerziellen Entwickler von der Firma viel direkter angesprochen werden konnten und zusätzlich zur Teilnahme ermahnt wurden.

9.1 Alter und Geschlecht

Die Open-Source-Programmierer sind im Durchschnitt 28.7 Jahre alt, während die Entwickler in den kommerziellen Software-Firmen mit durchschnittlich 33.7 Jahren rund 5 Jahre älter sind. Diese Altersdifferenz ist hochsignifikant ($\alpha < 1\%$) (siehe Tabelle 9.1).

Der Anteil der Open-Source-Programmiererinnen beträgt rund 2%, während im kommerziellen Umfeld der Anteil der Software-Entwicklerinnen mit 12% rund sechsmal höher ist (siehe Kreuztabelle 9.2). Der grössere weibliche Anteil im kommerziellen Umfeld ist statistisch hochsignifikant.

Tabelle 9.1: Altersvergleich

Altersvergleich		
	Mittel	Anzahl
Open-Source-Entwickler	28.72	1274
kommerzielle Software-Entwickler	33.65	113
Total	29.12	1387

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 9.2: Vergleich der Geschlechter-Verhältnisse

Geschlecht			
	Open-Source	kommerziell	Total
männlich	1270	99	1369
	(1259)	(110)	
	98.1%	87.6%	97.3%
weiblich	24	14	38
	(35)	(3)	
	1.9%	12.4%	2.7%
Total	1294	113	1407

Bemerkung: In Klammern die erwartete Anzahl

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

9.2 Vergleich der Flow-Komponenten

Wenn wir vergleichen wollen, ob die Open-Source-Programmierer häufiger Flow erleben bei ihrer Tätigkeit als Software-Entwickler in kommerziellen Software-Firmen, so stellt sich die Frage, welches Set der Flow-Komponenten für einen Vergleich herangezogen werden soll. Die Abbildungen 9.1 und 9.2 zeigen aber in beiden Fällen ein konsistentes Bild. In beiden Fällen weisen die Open-Source-Entwickler für die Faktoren „Konzentration“, „Flow/Spass“, „Eindeutigkeit der Aufgabe“ und „Zeitempfinden/Immersion“ die höheren Werte auf.

Ein T-Test für die Mittelwertgleichheit der Faktoren aus der Open-Source-Umfrage beweist, dass die Differenz im Faktor „Flow/Spass“ statistisch hochgradig signifikant ist (Signifikanzniveau $\alpha = 1\%$, siehe Tabelle 9.3). Aus diesem Set ist zusätzlich der Faktor „Zeitempfinden/Immersion“ ($\alpha = 10\%$) statistisch signifikant. Werden die Faktoren untersucht, die sich aus der Umfrage unter den kommerziellen Software-Entwicklern ergeben, so ist neben „Flow/Spass“ auch „Herausforderung“ hochsignifikant verschieden. In diesem Set ist zusätzlich der Unterschied im Faktor „Eindeutigkeit der Aufgabe“ signifikant ($\alpha = 5\%$).

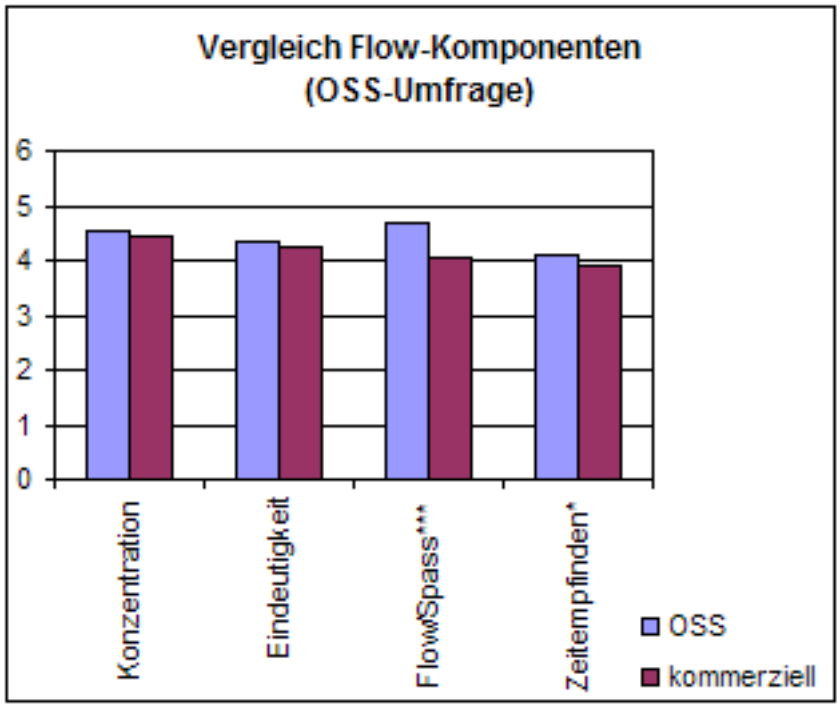
Aus dieser Analyse geht eindeutig hervor, dass die Software-Entwickler in einem kommerziellen Kontext weniger Spass bei ihrer Tätigkeit empfinden als ihre Kollegen im Open-Source-Umfeld. Die Open-Source-Entwickler taxieren die Möglichkeit, bei ihrer Tätigkeit Flow zu empfinden, mit rund einem halben Punkt höher als die kommerziellen Programmierer (auf der sechspunktigen Likert-Skala).

Interessant ist, dass die kommerziellen Software-Entwickler einzig bei der Komponente „Herausforderung“ einen signifikant höheren Wert aufweisen. Sichtbar spielt die Herausforderung für die Programmierer im kommerziellen Umfeld eine grössere Rolle als für Software-Entwickler in Open-Source-Projekten.

Bei diesem Vergleich stellt sich allerdings die Frage, ob der Unterschied nicht durch die eingangs erwähnten systematischen Effekte verursacht sein könnte. Um diese Möglichkeit auszuschliessen, müsste ich aus den Antworten der Open-Source-Umfrage zwei Mengen bilden können, wovon die eine den unter kommerziellen Bedingungen arbeitenden Software-Entwicklern entspricht, während die andere aus den Programmierern gebildet wird, die in ihrer Freizeit an Open-Source-Projekten arbeiten. Mit einem Vergleich von zwei solchen Gruppen könnte die Problematik von systematischen Effekten elegant umgangen werden.

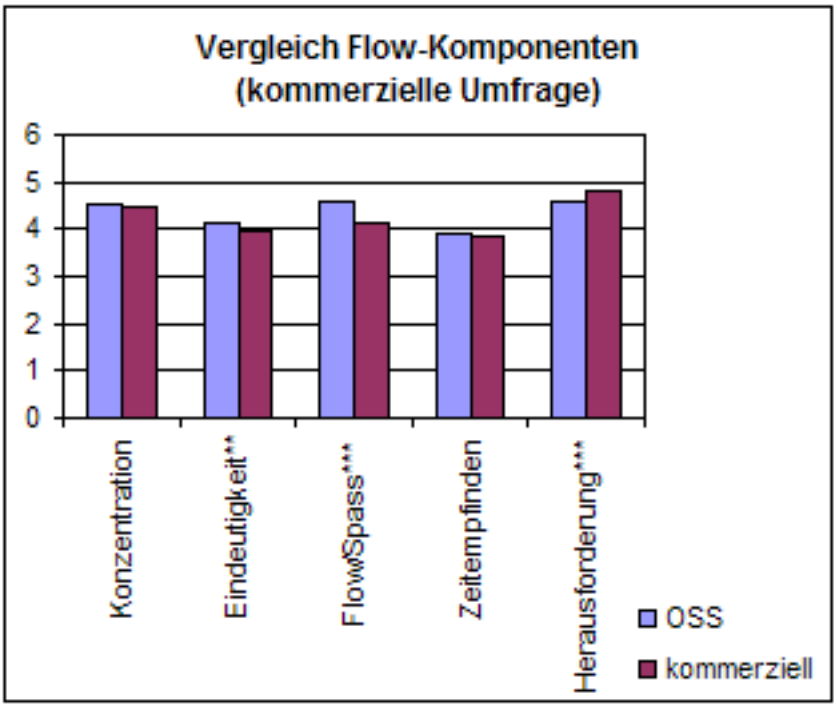
Diese Möglichkeit steht mir tatsächlich zur Verfügung. In Frage 41 des Open-Source-Fragebogens habe ich mich bei den Software-Entwicklern erkundigt, wie gross der Anteil des Zeiteinsatzes für Open Source ist, den sie in ihrer Freizeit leisten. Die Programmierer, die weniger als 10% ihres Engagements in der Freizeit, d.h. mehr als 90% während der Arbeitszeit leisten, können als *Professionals* bezeichnet werden, d.h. als Open-Source-Entwickler, die unter kommerziellen Bedingungen arbeiten. Bei den Entwicklern dagegen, deren Open-Source-Engagement zu mehr als 90% in der Freizeit stattfindet, handelt es sich zweifellos um die klassischen *Open-Source-Hacker*. Wie ich in Tabelle 7.18 gezeigt habe, gehören rund 12% (153 Personen) des Samples zu den Professionals und 40% (518 Programmierer) zu den Hackern.

Wenn ich nun das Flow-Empfinden der Professionals und Open-Source-Hacker vergleiche (siehe Abbildung 9.3) erhalte ich praktisch das gleiche Bild wie in Figur



Bemerkung: *** Korrelation auf 1% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Abbildung 9.1: Vergleich der Faktoren aus der OSS-Umfrage



Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant

Abbildung 9.2: Vergleich der Faktoren aus der kommerziellen Umfrage

Tabelle 9.3: Flow-Empfinden: Vergleich

T-Test für die Mittelwertgleichheit			
Flow-Faktor	Differenz	Signifikanz	T-Wert
Faktoren aus OSS-Umfrage			
Konzentration	0.111	0.15	1.439
Eindeutigkeit der Aufgabe	0.096	0.14	1.476
Flow/Spass	0.594***	0.00	8.896
Zeitempfinden/Immersion	0.164*	0.09	1.716
Faktoren aus kommerzieller Umfrage			
Konzentration	0.078	0.33	0.979
Eindeutigkeit der Aufgabe	0.179**	0.02	2.310
Flow/Spass	0.433***	0.00	5.784
Zeitempfinden/Immersion	0.096	0.30	1.046
Herausforderung	-0.238***	0.00	-3.325

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

9.1. Die Open-Source-Hacker zeigen in allen Komponenten die besseren Werte. Für die Komponenten „Konzentration“, „Flow/Spass“ und „Zeitempfinden/Immersion“ ist der Unterschied statistisch hochsignifikant (siehe Tabelle 9.4).

Dieses Resultat zeigt, dass es nicht das Studiendesign ist, welches die Unterschiede in den Flow-Komponenten zwischen kommerziellen und Open-Source-Entwicklern verursacht. Es bleibt noch einen Einwand zu entkräften, um aus diesem Resultat zu folgern, dass die nachgewiesenen Differenzen durch die Unterschiede in den Software-Entwicklungsmodellen verursacht sind. Die Unterscheidung in Professionals und Hacker habe ich im Nachhinein gemacht, auf Grund der Information über das zeitlichen Engagement während der Freizeit. Ich gehe dabei von der Vorstellung eines Software-Entwicklers mit bestimmten Fähigkeiten und Neigungen aus, welcher in einer bestimmten Lebensphase an Open-Source-Projekten überwiegend als Hacker in der Freizeit arbeitet und in einer anderen Phase als Professional überwiegend für diese Arbeit bezahlt wird. *Mutatis mutandis* wird dieser Programmierer, auf Grund der Differenzen zwischen den Entwicklungsmodellen, unter kommerziellen Bedingungen weniger Spass am Programmieren haben.

Nun könnte es aber sein, dass der Open-Source-Hacker ein Typ Programmierer *a priori* ist. Diese Typ würde sich durch seine grosse Freude am Programmieren von Open-Source-Software auszeichnen und wäre überdurchschnittlich häufig in der Population der Software-Entwickler vertreten, die hauptsächlich in der Freizeit an Open-Source-Projekten arbeiten. Damit würde der festgestellte Unterschied beim Empfinden von Spass nicht durch die Unterschiede in den Entwicklungsmo-

Tabelle 9.4: Flow-Faktoren: Open-Source-Hacker und Professionals

T-Test für die Mittelwertgleichheit			
Flow-Faktor	Differenz	Signifikanz	T-Wert
Konzentration	0.360^{***}	0.00	4.266
Eindeutigkeit der Aufgabe	0.050	0.59	0.533
Flow/Spass	0.263^{***}	0.00	3.395
Zeitempfinden/Immersion	0.277^{***}	0.01	2.724

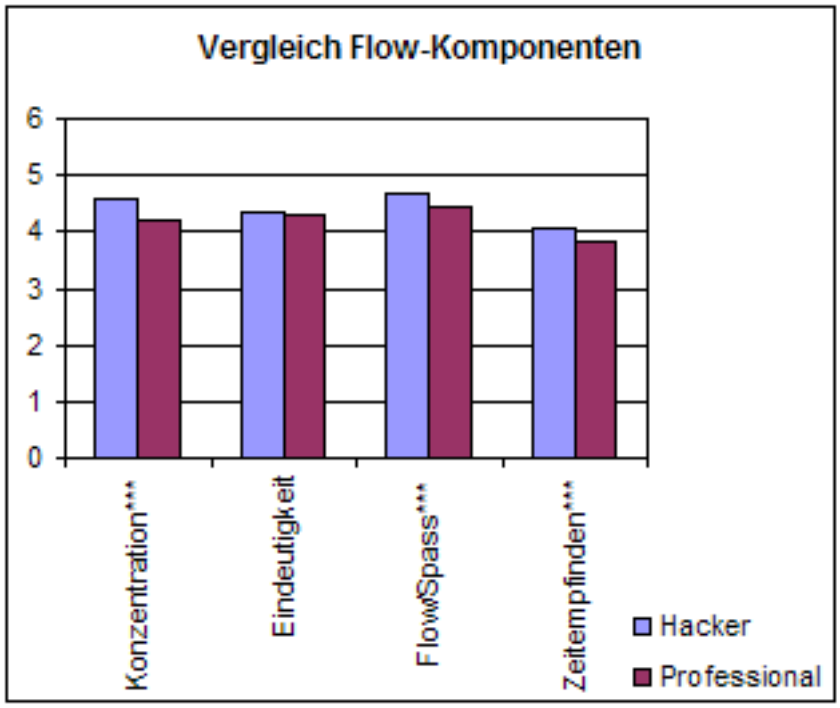
Bemerkung: *** Korrelation auf 1% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

dellen, sondern durch die Verschiedenheit der Entwickler-Typen erklärt.

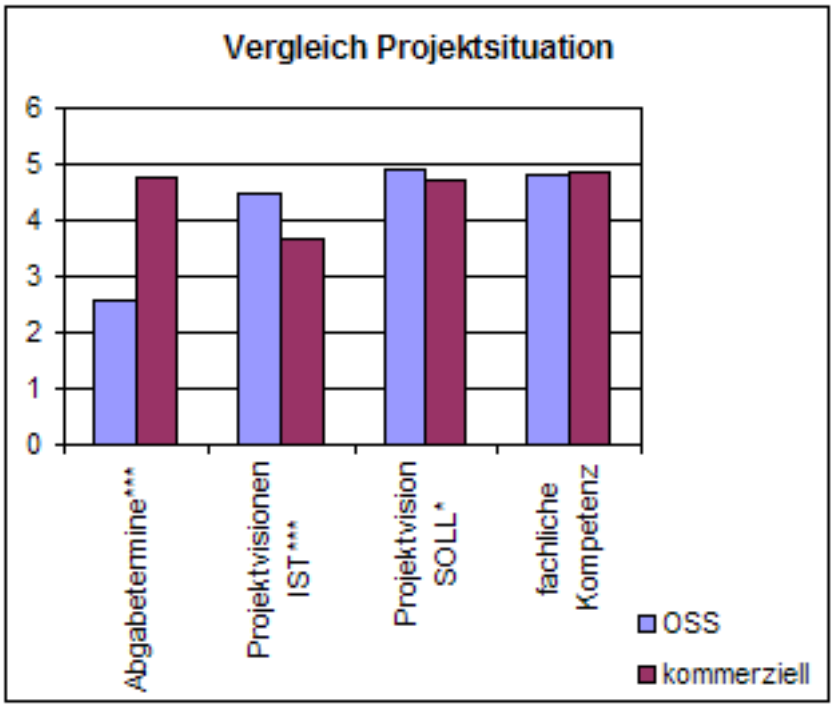
Dieser Einwand ist nur dann stichhaltig, wenn ich einen Filtermechanismus nachweisen kann, welcher dafür sorgt, dass der hypothetisch angenommene Entwickler-Typ „Hacker“ vorwiegend in der Population der Software-Entwickler auftaucht, welche hauptsächlich in der Freizeit an Open-Source-Projekten arbeiten, dagegen nur selten in der Population der bezahlten Open-Source-Programmierer anzutreffen ist. Wie könnte man sich einen Software-Entwickler vorstellen, der speziell durch die Kombination „Open Source“ und „Programmieren in der Freizeit“ angelockt wird? Programmieren als Tätigkeit unterscheidet sich in keiner Weise, ob diese Aktivität in der Freizeit oder während der Arbeitszeit ausgeübt wird. Was sich ändert ist das Umfeld, d.h. das Entwicklungsmodell. Die gleiche Argumentation gilt auch bezüglich der Software-Lizenz: Ob eine Applikation unter einer Open-Source-Lizenz freigegeben oder kommerziell vertrieben wird, ändert an der Tätigkeit des Programmierens, abgesehen vom Entwicklungsmodell, nichts. Dies zeigt auch ein Blick auf SourceForge. Unter einer Open-Source-Lizenz wird die ganze Bandbreite von möglichen Software-Applikationen angeboten. Der Typ der Lizenz sagt nichts über die Art der Software aus, deshalb kann der Typ der Lizenz auch keinen speziellen Typ von Programmierer anlocken. Aus diesem Grund bin ich der Ansicht, dass es nur die Merkmale der Open-Source-Entwicklungsmodells sein könnten, welche einen Spass-getriebenen Programmierer dazu bringen, unbezahlt an Open-Source-Projekten zu arbeiten.

Ich gehe deshalb davon aus, dass die festgestellten signifikanten Unterschiede in Tabelle 9.3 weder durch systematische Effekte noch durch einen speziellen Entwickler-Typ „Open-Source-Hacker“ verursacht sind, sondern tatsächlich die Bedingungen des kommerziellen Umfelds im Vergleich zum Software-Entwickeln im Open-Source-Kontext widerspiegeln. Die Resultate dieses Abschnitts bestätigen somit die Hypothese, dass die kommerziellen Bedingungen den Spass am Programmieren negativ beeinflussen.



Bemerkung: *** Korrelation auf 1% Niveau signifikant

Abbildung 9.3: Flow-Faktoren: Open-Source-Hacker und Professionals



Bemerkung: *** Korrelation auf 1% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Abbildung 9.4: Projektsituation: Vergleich OSS und kommerzielle Umfrage

9.3 Vergleich der Projektsituation

Neben den Flow-Faktoren wurde auch nach der Projektsituation (Projektvision, Abgabetermine, fachliche Kompetenz des Projektleiters) in beiden Umfragen auf analoge Art gefragt. Ein Vergleich der Antworten auf diese Fragen zeigt klar die Unterschiede zwischen dem Open-Source-Entwicklungsmodell und dem Arbeitsumfeld bei kommerziellen Software-Projekten. Wie erwartet spielen Abgabetermine im Kontext von Open-Source-Projekten praktisch keine Rolle, während sie in kommerziellen Projekten einen durchaus prägenden Faktor darstellen (siehe Abbildung 9.4 und Tabelle 9.5).

Hochsignifikant ist auch der Unterschied bei der Bewertung der Projektvision. Eine das Software-Projekt tragende Vision wird bei Open-Source-Projekten viel deutlicher verspürt als bei Software-Projekten im kommerziellen Umfeld. Viel weniger ausgeprägt ist dagegen der Unterschied bezüglich der Wichtigkeit, die einer Projektvision zugeschrieben wird. Die Abbildung zeigt, dass die kommerziellen Software-Entwickler einer Projektvision etwas weniger Wert beimessen, wobei diese Differenz nur noch auf dem 10%-Niveau signifikant ist.

Wieder will ich prüfen, ob die festgestellten Unterschiede frei von systematischen Effekten sind, indem ich die Antworten der Open-Source-Hacker mit de-

Tabelle 9.5: Projektsituation: Vergleich OSS und kommerzielle Umfrage

T-Test für die Mittelwertgleichheit			
Projektsituation	Differenz	Signifikanz	T-Wert
Abgabetermine	-2.203***	0.00	-21.311
Projektvision IST	0.826***	0.00	7.010
Projektvision SOLL	0.211*	0.05	1.935
fachliche Kompetenz	-0.038	0.72	-0.357

Bemerkung: *** Korrelation auf 1% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Tabelle 9.6: Projektsituation: Vergleich Hacker und Professionals

T-Test für die Mittelwertgleichheit			
Projektsituation	Differenz	Signifikanz	T-Wert
Abgabetermine	-0.779***	0.00	-5.573
Projektvision IST	0.354***	0.01	2.780
Projektvision SOLL	-0.060	0.57	-0.570
fachliche Kompetenz	-0.213*	0.06	-1.898

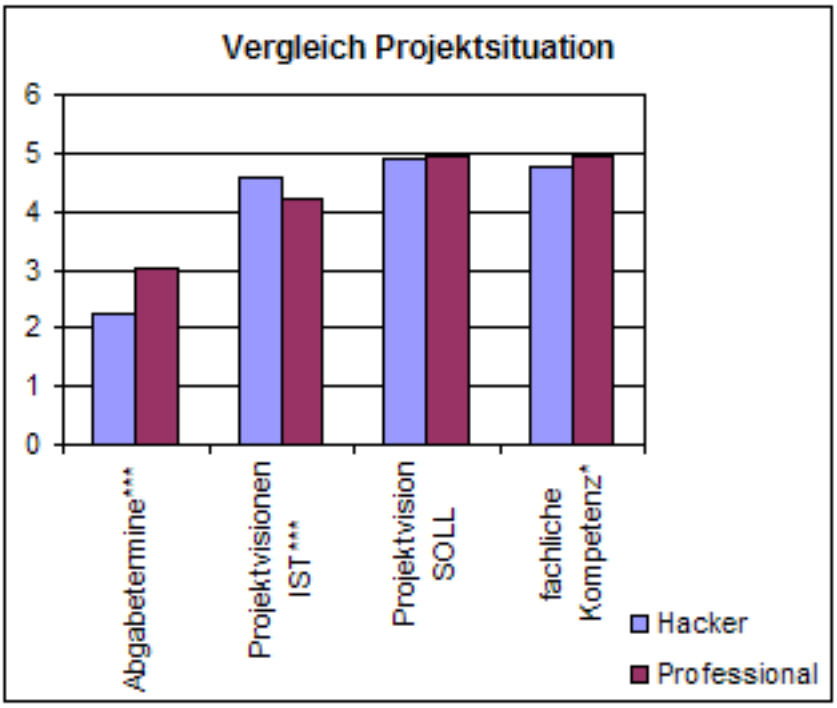
Bemerkung: *** Korrelation auf 1% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

nen der Professionals vergleiche, die alle aus der Open-Source-Umfrage kommen. Auch diese Gegenüberstellung bestätigt das Bild, welches aus dem Vergleich der Open-Source-Entwickler und der kommerziellen Programmierer gewonnen wurde. Wieder sind die Unterschiede bei den Abgabeterminen und dem Ist-Zustand von Projektvisionen statistisch hochsignifikant (Tabelle 9.6) und wieder spielen Abgabetermine für Professionals eine grössere Rolle als für Open-Source-Hacker, während Professionals häufiger in Projekten arbeiten, die weniger Wert auf eine Projektvision legen (Abbildung 9.5).

Bemerkenswert ist, dass die Teilnehmer an der kommerziellen Umfrage einen wesentlich grösseren Wert angegeben haben bei der Spürbarkeit von Abgabeterminen als die Professionals, welche die Open-Source-Umfrage beantwortet haben, wohingegen die Werte bei der Spürbarkeit von Projektvisionen sehr ähnlich sind. Ebenfalls interessant ist die Tatsache, dass der Unterschied bei der Wertschätzung der fachlichen Kompetenz des Projektleiters beim zweiten Verglich signifikant ist, allerdings nur auf dem 10%-Niveau.

Diese Gegenüberstellungen machen deutlich, dass der Projektvision im Open-



Bemerkung: *** Korrelation auf 1% Niveau signifikant
* Korrelation auf 10% Niveau signifikant

Abbildung 9.5: Projektsituation: Vergleich Hacker und Professionals

Tabelle 9.7: Projektvision: Diskrepanz zwischen Soll- und Ist-Zustand

T-Test für die Mittelwertgleichheit			
Projektvision	Differenz	Signifikanz	T-Wert
Vergleich Umfragen	0.622^{***}	0.00	3.728
Vergleich Hacker vs. Professionals	0.359^{***}	0.01	2.709

Bemerkung: *** Korrelation auf 1% Niveau signifikant

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Source-Umfeld eine viel wichtigere Rolle zukommt als wenn unter kommerziellen Bedingungen gearbeitet wird. Bemerkenswert ist aber, dass die Diskrepanz zwischen Soll- und Ist-Zustand bei den kommerziellen Entwicklern viel deutlicher verspürt wird, wie aus der Tabelle 9.7 abzulesen ist. Offensichtlich ist der Wunsch nach einer Projektvision auch unter den Programmieren in kommerziellen Software-Projekten oder unter kommerziellen Bedingungen in weit grösserem Mass vorhanden, als dass er im Projekt-Alltag erfüllt wird.

9.4 Analyse der Unterschiede

Die Resultate im Abschnitt 9.2 bestätigen meine Forschungs-Hypothese 3: Die gleiche Tätigkeit, das Entwickeln von Software, macht, wenn sie im Rahmen eines Open-Source-Projekts ausgeführt wird, signifikant mehr Spass als wenn sie unter kommerziellen Bedingungen betrieben wird. Damit stellt sich natürlich die Frage, welche Randbedingung für diesen Unterschied verantwortlich ist (Forschungsfrage 4). Welches Merkmal des Open-Source-Entwicklungsmodells ist der Grund für die Tatsache, dass Programmieren in einem Open-Source-Projekt mehr Freude macht? Und wenn eine solche Ursache identifiziert werden kann: Ist dieses Merkmal konstituierend für das Open-Source-Entwicklungsmodell oder kann es prinzipiell auch im kommerziellen Umfeld zur Wirkung gebracht werden?

In Kapitel 3.6 (Tabelle 3.2) habe ich einige Merkmale des Open-Source-Entwicklungsmodells identifiziert, die Auswirkungen auf das Programmieren haben und so zu Unterschieden führen können, je nachdem, ob diese Tätigkeit unter kommerziellen oder Open-Source-Bedingungen ausgeübt wird. In der Tabelle 9.8 habe ich diese charakteristische Differenzen nochmals gegenübergestellt.

Open-Source-Projekte[1] sind von einer starken *Projektvision* getragen. Der Projekt-Eigentümer weiss, warum er das Projekt lanciert und unter einer Open-Source-Lizenz freigegeben hat. Er hat genaue Vorstellungen über die Ziele seines Projekts. Auch hat er gute Gründe, die Projekt-Ziele anderen Personen, z.B. den aktuellen und potentiellen Programmierern im Projekt zu vermitteln: Eine überzeugende

[1] In dieser Diskussion verstehe ich unter Open-Source-Projekten solche Projekte, die auf Grund von freiwilliger Zusammenarbeit zustande kommen. Die Bemerkungen und Folgerungen gelten für Open-Source-Projekte, an denen bezahlte Beitragsleister arbeiten, in geringerem Mass.

Tabelle 9.8: Unterschiede zwischen den Software-Entwicklungsmodellen

Merkmal	**OSS**	**kommerziell**
Projektvision	+	?
formale Autorität	-	+
monetäre Anreize	-	+
Abgabetermine	-	+
optimale Herausforderung	+	?

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Projektvision ist das geeignete Mittel, potentielle Beitragsleister zu einem Engagement zu bewegen sowie die Beiträge der aktuell am Projekt beteiligten Software-Entwickler zu koordinieren und auf das gemeinsame Ziel auszurichten. Projektvisionen können auch in kommerziellen Software-Projekten eine Rolle spielen. Allerdings ist in erster Linie die formale Autorität der hierarchisch vorgesetzten Person dafür verantwortlich, ob ein Programmierer in diesem oder jenem Software-Projekt mitmacht und welche Arbeiten er konkret ausführt. Der Projektvision kommt deshalb nicht die gleiche, koordinierende Rolle zu wie bei Open-Source-Projekten.

Die hierarchische Organisation einer Firma liefert ein weiteres Unterscheidungsmerkmal. Hierarchien begründen *formale Autorität*. Die Inhaber formaler Autorität können bei ihren Mitarbeitenden ein bestimmtes Verhalten anordnen, was den Projekt-Verantwortlichen im Open-Source-Umfeld nicht möglich ist. Letztere müssen mit Hilfe ihrer fachlichen Kompetenz und auf Grund einer sachlichen Auseinandersetzung versuchen, eine Verhaltensänderung zu bewirken, wenn sie mit den Beiträgen der am Open-Source-Projekt mitwirkenden Programmierer nicht zufrieden sind.

Neben Hierarchie und formaler Autorität sind die *monetären Anreize*, welche eine Firma setzen kann, ein spezifisches Merkmal des professionellen Arbeitsumfelds: Jede Beitragsleistung wird entlohnt. Diese Möglichkeit steht einem Open-Source-Projekt, welches auf einer freiwilligen Zusammenarbeit der Beteiligten beruht, nicht zur Verfügung.

Kommerzielle Software-Projekte sind üblicherweise in einen kommerziellen Produktions- und Verwertungs-Prozess eingebettet: Die Software muss zu einem bestimmten Termin mit einer definierten Funktionalität zur Verfügung stehen. *Abgabetermine* sind unvermeidlicher Bestandteil von kommerziellen Produktions-Abläufen. In einem Open-Source-Kontext dagegen kann jeder Beteiligte am Projekt problemlos einem vom Projekt-Eigentümer gesetzten Abgabetermin ausweichen. Wie jeder andere Aspekt der freiwilligen Zusammenarbeit kommen auch Abgabetermine nur auf Grund einer freiwilligen Übereinkunft zustande.

Ein letztes Unterscheidungsmerkmal zwischen den Software-Entwicklungsmodellen besteht bezüglich der *Herausforderung* an den Software-Entwickler. Bei einem Open-Source-Projekt kann der Programmierer die Herausforderung durch die freie Wahl des Projekts, an welchem er sich beteiligen will, selbst steuern. Eine

optimale Herausforderung kann somit als gegeben betrachtet werden. Im kommerziellen Umfeld ergibt sich ein Projekt-Engagement dagegen aus den betrieblichen Umständen. Auf das Potential, welches der Programmierer mit seinem technischen Wissen und seinen Erfahrungen darstellt, sowie seinen Wünschen bezüglich fachlicher und persönlicher Entfaltung kann das Projekt nur in beschränktem Mass Rücksicht nehmen.

Wie kann nun geprüft werden, ob diese Unterscheidungsmerkmale für die festgestellte Differenz im Empfinden von Flow beim Programmieren verantwortlich sind? Wenn ein Merkmal eine Wirkung auf das Flow-Empfinden hat, so sollte sich eine Korrelation des Merkmals mit den Flow-Faktoren feststellen lassen. Bei den Abgabeterminen erwarten wir beispielsweise, dass die Programmierer umso weniger Spass empfinden, je mehr sie unter Abgabeterminen zu leiden haben. Ebenso sollte die Freude an der Tätigkeit leiden, je mehr die formale Autorität im Projekt spürbar ist. Umgekehrt erwarten wir einen positiven Effekt auf den Spass, je besser die Projektvision spürbar ist und je optimaler die Herausforderung des Projekts an das Potential des Entwicklers angepasst ist. Welche Auswirkungen monetäre Anreize auf die Freude an der Arbeit des Software-Programmierens hat, lässt sich apriori nicht voraussagen. Aus nachvollziehbaren Gründen des Datenschutzes war es mir nicht möglich, Informationen über das Einkommen der Programmierer in den Software-Firmen zu erheben. Aus diesem Grund ist es mit nicht möglich, die Wirkung von monetären Anreizen auf das Erleben von Flow zu untersuchen.

Zum Testen der Hypothesen können die Antworten auf die Fragen betreffend der Projektsituation, die ich in der kommerziellen Umfrage erhoben habe, verwendet werden. Als Mass für die optimale Herausforderung wurde ein Index[2] aus den Variablen 33 und 35 aus dem Abschnitt betreffend das Verhältnis zum Arbeitgeber gebildet. Die Tabelle 9.9 zeigt die Liste der signifikanten Korrelationen mit ihren Vorzeichen.[3]

Interessant ist, dass das Merkmal „Abgabetermine“ zwar hochsignifikante Korrelationen zeigt, entgegen den Erwartungen erscheinen die Korrelationskoeffizienten aber mit einem positiven Vorzeichen. Das bedeutet, dass die Software-Entwickler im kommerziellen Umfeld umso mehr Flow und Spass empfinden, je spürbarer der Druck von Abgabeterminen ist. Dies widerspricht den Erfahrungen von DeMarco und Lister (1987) und Amabile u. a. (1976), welche eine nachteilige Auswirkung von Abgabeterminen auf die intrinsische Motivation festgestellt haben (siehe Abschnitt 3.6). Gemäss meinen Daten wirkt sich der Druck von Abgabeterminen nicht nur auf das Empfinden von Spass positiv aus, auch das Versinken in die Programmier-Tätigkeit wird hochsignifikant gefördert, mit positiven Auswirkungen auf die Konzentration und Eindeutigkeit der Aufgabe.

Den Erwartungen entsprechend positiv korreliert eine spürbare Projektvision mit der Freude an der Tätigkeit. Plausibel ist, dass sich eine nachvollziehbare Projektvision positiv auf die Eindeutigkeit der Aufgabe im Alltag auswirkt. Dies hat sicher auch einen positiven Effekt auf die Konzentration, mit der die Programmierer ihrer Tätigkeit nachgehen können. Interessant ist allerdings, dass schon der

[2] Frage 33: „Ich kann mich bei der Firma persönlich und beruflich weiterentwickeln“, Frage 35: „Bei der Firma werden das Rollenmodell und die Prozesse professionell und erfolgreich umgesetzt“.

[3] Der Faktor „Herausforderung“ zeigt keine signifikanten Korrelationen und wurde aus der weiteren Analyse ausgeschlossen.

Tabelle 9.9: Korrelationen: Unterscheidungsmerkmale mit Flow-Faktoren

	Flow-Faktoren			
Merkmal	Flow/ Spass	Konzentration	Zeitempfinden/ Immersion	Eindeutigkeit
38 Abgabetermine	0.256***	0.228**	0.278***	0.178*
39 Projektvision: IST	0.397***	0.322***	0.155	0.384***
41 Projektvision: SOLL	0.189**	0.037	0.046	-0.175*
40 formale Autorität	0.175*	0.179*	0.117	0.139
42 fachliche Kompetenz	-0.071	-0.015	-0.177*	0.012
optimale Herausforderung	0.406***	0.208**	0.220**	0.347***

Bemerkung: *** Korrelation auf 1% Niveau signifikant
** Korrelation auf 5% Niveau signifikant
* Korrelation auf 10% Niveau signifikant
n = 107 bis 112

Quelle: Benno Luthiger, FASD Studie, Universität Zürich

Wunsch nach einer Projektvision signifikant mit der empfundenen Freude an der Tätigkeit korreliert.

Nur schwach signifikante Korrelationen weisen die Merkmale „formale Autorität“ und „fachliche Kompetenz“ auf. Darüber hinaus besitzen die Korrelationskoeffizienten das falsche Vorzeichen. Wir hätten verminderten Spass erwartet, je spürbarer die formale Autorität ist.

Hingegen entspricht der signifikant positive Zusammenhang der optimalen Herausforderung mit dem Empfinden von Flow unseren Erwartungen. Gelingt es einem Arbeitgeber, für die Software-Entwickler optimale Arbeitsbedingungen zu bieten, so wirkt sich das für die Eindeutigkeit der Aufgabe, das Versinken in die Tätigkeit und die Konzentration ebenfalls sehr vorteilhaft aus.

Damit können die Forschungsfragen 3 und 4 auf Grund dieser Untersuchungen wie folgt beantwortet werden: Das Entwickeln von Software macht signifikant mehr Spass, wenn diese Tätigkeit im Kontext eines Open-Source-Projekts ausgeübt wird als wenn sie unter kommerziellen Bedingungen erfolgt. Der Grund für diesen Unterschied sind aber nicht die Abgabetermine, welche auf Grund der Produktions-Bedingungen unvermeidliche Bestandteile von kommerziellen Software-Projekten sind. Das Vorhandensein von Abgabeterminen müsste im Gegenteil dazu führen, dass die Programmierer im kommerziellen Bereich mehr und nicht weniger Spass haben. Vielmehr können die Unterschiede bei der Projektvision und der optimalen Herausforderung die Differenz beim Flow-Empfinden erklären. Wie die Abbildungen 9.4 und 9.5 (und die Tabellen 9.5 und 9.6) zeigen, sind Projektvisionen bei Open-Source-Projekten signifikant deutlicher spürbar. Gleichzeitig korrelieren nachvollziehbare Projektvisionen signifikant positiv mit dem Spass an der Tätigkeit. Ein Unterschied in der Projektvision macht sich somit direkt bemerkbar im Spass am Programmieren. Die gleiche Argumentation gilt auch bezüglich der

optimalen Herausforderung.

9.5 Folgerungen

In diesem Kapitel habe ich das Flow-Empfinden von Open-Source-Entwicklern mit der Freude am Programmierern bei kommerziellen Programmierern verglichen. Zur Vermeidung von systematischen Verzerrungen habe ich diesen Vergleich auch zwischen Open-Source-Hackern und professionellen Open-Source-Entwicklern durchgeführt. Das Ergebnis dieser Untersuchungen zeigt zweifelsfrei, dass Programmieren mehr Spass macht, wenn es im Kontext eines Open-Source-Projekts durchgeführt wird. Auf die Frage nach den Gründen dafür, dass das Open-Source-Entwicklungsmodell den Software-Entwicklern bessere Möglichkeiten gibt, Spass am Programmieren zu haben, habe ich zwei Ursachen identifizieren können: Open-Source-Projekte werden durch eine Projekt-Vision gestützt und sie ermöglichen es den Entwicklern, eine ihren Fähigkeiten angepasste Herausforderung zu finden.

Schon in meinen Schlussfolgerungen in Kapitel 8.5 habe ich darauf hingewiesen, dass sich für einen Arbeitgeber Investitionen in die Projektvision und in eine Optimierung der Übereinstimmung von Projekt-Herausforderungen und Programmier-Fähigkeiten auszahlen könnten. Solche Massnahmen steigern den Spass der Software-Entwickler und auf diese Weise, wie ich abgeleitet habe, gleichzeitig deren Produktivität. Wenn ein Arbeitgeber offen für unkonventionelle Schritte ist, um die Produktivität seiner Software-Entwickler zu heben, müsste da nicht der Ratschlag lauten, er solle gleich das ganze Open-Source-Entwicklungsmodell übernehmen (mit Ausnahme der Lizenz)?

Ich will dieser Frage nachgehen, indem ich sie umkehre und danach Frage, welche Elemente des Open-Source-Entwicklungsmodells überhaupt in einen kommerziellen Kontext transferiert werden können. In Tabelle 3.1 habe ich die Komponenten dieses Entwicklungsmodells aufgeführt. Nun will ich für jede dieser Komponenten die Frage stellen: Welche Bedeutung hat sie für den Spass am Programmieren und wie könnte dieses Element unter kommerziellen Bedingungen realisiert werden?

Die personell offene Entwicklergemeinschaft eines Open-Source-Projekts kann für ein kommerzielles Software-Projekt insofern nicht realisiert werden, als dass die Programmierer im kommerziellen Kontext für ihre Arbeit entschädigt werden und deshalb auf irgendeine Weise in einer vertraglich festgelegte Beziehung zum Arbeitgeber stehen. Der Umstand, dass die Gruppe der am Projekt beteiligten Programmierer offen oder geschlossen ist, dürfte aber auf das Flow-Empfinden der Software-Entwickler keinen Einfluss haben.

Das Merkmal, dass die Entwickler in Open-Source-Projekten meistens auch die Benützer der Applikation sind, dürfte in kommerziellen Projekten nur in Ausnahmefällen zutreffen. Dieser Umstand hat Auswirkungen auf die Projekt-Vision. Für Prosumer ergibt sich die Projekt-Vision unmittelbar aus ihren Bedürfnissen. Wo die Benützer nicht identisch sind mit den Produzenten, erhöht sich der Bedarf an einer expliziten und nachvollziehbaren Projekt-Vision.

Die Absenz formaler Hierarchien in Open-Source-Projekten lässt sich für kommerzielle Projekte ebenfalls nicht realisieren. Zweifellos lassen sich aber die nega-

tiven Auswirkungen solcher Hierarchien mildern, wenn ein Unternehmen durch flache Hierarchien geprägt ist und diese Hierachien gebildet werden durch transparente, durch meritökonomische Mechanismen gesteuerte Prozesse. Wie meine Auswertungen zudem zeigen (siehe Tabelle 9.9), tut das Vorhandensein von formaler Autorität der Freude am Programmieren keinen Abbruch.

Open-Source-Projekte zeichnen sich durch eine grosse Releasehäufigkeit aus. Für einen Software-Entwickler ist es befriedigend, wenn er sieht, dass sein Beitrag rasch als Bestandteil der Applikation erscheint und nicht z.B. vom Projektleiter auf Grund mangelnder Qualität entfernt oder überarbeitet werden muss. Dies ist eine Frage des Qualitätsmanagements des Software-Projekts und damit unabhängig davon, ob es sich um ein kommerzielles oder Open-Source-Projekt handelt.

Das Merkmal der Freiwilligkeit der Beitragsleistung in Open-Source-Projekten lässt sich in kommerziellen Projekten, wo die Entwickler für ihre Arbeit bezahlt werden, nicht umsetzen. Die Freiwilligkeit ist aber das entscheidende Element, wenn es darum geht, die optimale Herausforderung für den Programmierer bei gegebenen Fähigkeiten zu finden. Die Freiwilligkeit der Beitragsleistung führt dazu, dass es die Programmierer sind, welche sich das Projekt auswählen, in welchem sie sich engagieren wollen. Ein ausschlaggebender Faktor für die Entscheidung des Programmierers dürfte die Herausforderung sein, welche ihm das Software-Projekt verspricht.[4] Der Programmierer benötigt dabei keine vollständigen Informationen, weder über das Projekt noch über seine Fähigkeiten. Die Freiwilligkeit der Teilnahme erlaubt es ihm, über ein *trial and error*-Verfahren innert kürzester Zeit jenes Projekt zu finden, das seinen aktuellen Fähigkeiten optimal entspricht. Dies ist möglich, weil es einen Markt gibt: Open-Source-Projekte buhlen um das Engagement von beitragswilligen Software-Entwickler und diese können sich das ihnen passende Projekt aussuchen und nach Belieben wieder verlassen.

Diese Art von Freiwilligkeit lässt sich für kommerziell arbeitende und entlohnte Programmierer nicht realisieren. Auf diesen Sachverhalt weist auch Barnett in ihrem Artikel über die Anwendbarkeit des Open-Source-Prozesses in der kommerziellen Software-Entwicklung hin (Barnett, 2004, S. 8f).[5] Wäre es aber möglich, innerhalb des Software-Unternehmens einen Marktplatz einzurichten, auf dem die verschiedenen Projekte der Firma um das Engagement der angestellten Software-Entwickler kämpften? In einem solchen Modell wären die Programmierer der Firma zwar verpflichtet, Software zu entwickeln, hätten aber die Freiheit zu entscheiden, für welches Projekt sie arbeiten wollen. Dies könnte selbstverständlich nur funktionieren, wenn die Firma genügend gross wäre, um überhaupt eine Auswahl an Projekten anbieten zu können. Ein solches Modell würde für die meisten Firmen wohl einen radikalen Wechsel bedeuten von einer Output-zentrierten zu einer Mitarbeiter-fokussierten Vorgehensweise und damit den Programmierern ein seltenes Mass an Autonomie und Selbstverantwortung einräumen. Ein auf diese Weise

[4] Weitere mögliche Auswahlkriterien sind die Reputation von Applikationen (gewisse Programme mögen *cooler* bzw. *geekier* erscheinen) sowie, wieviel der Programmierer durch sein Engagement in einem bestimmten Projekt lernen kann.

[5] In ihrem Essay sieht Barnett keine Möglichkeit, eine freie Projektwahl für die angestellten Programmierer in einer kommerziellen Software-Firma zu realisieren. Sie weist aber auf die positiven Erfahrungen hin, die Firmen gewonnen haben, die ihren Entwicklern einen garantierten Freiraum (z.B. 10% oder 20% der Zeit) für vollkommen frei gewählte Arbeiten einräumen.

ausgestalteter Marktplatz käme aber dem Open-Source-Entwicklungsmodell sehr nahe und würde bezüglich Spass und Produktivität mit Sicherheit hervorragend abschneiden. Ob damit auch eine optimale Allokation der im Unternehmen angestellten Software-Entwickler erreicht und ob eine solche Software-Firma überhaupt effizient gesteuert werden könnte, diesen Fragen nachzugehen ist mir an dieser Stelle nicht möglich.

Kapitel 10

Schlussfolgerungen

Die Auswertungen in Kapitel 7.1.4 ergeben den Eindruck, dass es sich bei Open Source um ein Phänomen handelt, das sich vorwiegend in der Freizeit abspielt. Von den im Durchschnitt 12.6 Stunden pro Woche, welche die Software-Entwickler für Open-Source-Projekte aufwenden, fallen 58% in die Freizeit der Programmierer. Von den 1330 Software-Entwicklern, die an der Open-Source-Umfrage teilgenommen haben, arbeiten rund zwei Drittel ganz oder vorwiegend unbezahlt an Open-Source-Projekten. Mit grosser Wahrscheinlichkeit wird aber der professionelle Anteil an Open Source (42% der aufgewendeten Zeit, 33% der involvierten Software-Entwickler) unterschätzt. Die angeschriebenen Open-Source-Plattformen eignen sich hervorragend als Sammelbecken für Open-Source-Projekte aus dem Freizeitbereich. Professionelle Open-Source-Projekte können sich eine eigene Projekt-Infrastruktur leisten und sind aus diesem Grund in dieser Umfrage mit Sicherheit untervertreten. Auf Grund dieser Überlegung vermute ich, dass der professionelle und bezahlte Beitrag zu Open Source mit dem freiwilligen und unbezahlten gleichgezogen oder diesen möglicherweise schon überholt hat.

In Kapitel 7.5 habe ich zeigen können, dass Spass eine bedeutende Rolle spielt, um das Engagement von Open-Source-Entwicklern zu begründen. Mit der Freude am Programmieren kann man 27% der Einsatzbereitschaft erklären. Wird darüber hinaus die verfügbare Freizeit berücksichtigt, so kann damit 33% des Engagements für Open Source begründet werden. Interessant ist, dass Spass auch bei professionellen Open-Source-Entwicklern sowie bei Programmierern aus dem kommerziellen Umfeld eine wichtige Rolle spielt, um deren Einsatzbereitschaft zu motivieren.

Ein Vergleich der Daten aus der Open-Source-Umfrage mit denjenigen der Befragung der kommerziellen Software-Entwickler (siehe Kapitel 9) zeigt weiter, dass Programmierer in Open-Source-Projekten wie vermutet deutlich mehr Spass am Programmieren haben als Entwickler, die unter kommerziellen Bedingungen arbeiten. Das gleiche Ergebnis erhalte ich, wenn ich die Antworten der Freizeitprogrammierer (*Open-Source-Hacker*) mit den professionell an Open-Source-Projekten beteiligten Software-Entwicklern vergleiche. Dieser Unterschied kann demnach nicht durch systematische Verzerrungen des Forschungsdesigns erklärt werden, sondern ist ein Merkmal der Produktionsbedingungen, unter welchen Software-Entwickler arbeiten.

Diese Differenzen sind aber nicht durch den Druck von Abgabeterminen oder

die Existenz von formalen Hierarchien verursacht, wie ich im Abschnitt 9.4 zeigen konnte. Der Grund dafür, dass Programmieren im Kontext von Open Source mehr Spass macht, könnte vielmehr darin liegen, dass die Software-Entwickler in Open-Source-Projekten eine glaubwürdigere Projekt-Vision und vor allem eine ihren Bedürfnissen besser entsprechende Herausforderung vorfinden.

Die Resultate dieser Untersuchung vermögen den Eindruck von Open Source als das rätselhafte Produkt von selbstlosen, möglicherweise leicht verrückten Open-Source-Hackern zu einem Grossteil korrigieren. Software-Entwickler, die einen *homo-ludens-payoff* konsumieren wollen, handeln durchaus rational, wenn sie solche Software-Projekte und Projekt-Situationen suchen, die möglichst viel Spass versprechen. Wie meine Daten zeigen, bieten Open-Source-Projekte mehr und bessere Möglichkeiten, bei der Programmier-Tätigkeit Spass zu empfinden. Open-Source-Software kann demnach, zumindest zu einem gewissen Teil, verstanden werden als das Nebenprodukt einer Handlung, die eminent Spass macht, und eines Entwicklungsmodells, welches das Bedürfnis nach Spass optimal unterstützt.

Der Befund, dass weder Abgabetermine noch formale Hierarchien die Ursache sind, dass das Software-Entwickeln unter kommerziellen Bedingungen weniger Spass macht, lässt interessante Ableitungen zu. Sowohl Abgabetermine wie auch formale Hierarchien sind Bestandteile des professionellen und kommerziellen Alltages, welche nicht eliminiert werden können. Wenn sich Abgabetermine und Hierarchien negativ auf den Spass am Programmieren auswirken würden, dann gäbe es keine Chance, dann wäre das Entwickeln von Software im kommerziellen Umfeld notwendigerweise mit einem Verlust an Freude an der Tätigkeit verknüpft.

Da genau dies aber nicht der Fall zu sein scheint, sondern im Gegenteil sowohl Abgabetermine wie auch formale Hierarchien sich auf den Spass am Programmieren eher positiv auswirken, gibt es prinzipiell keinen Grund, dass Programmieren unter kommerziellen Bedingungen nicht im gleiche Mass Spass machen sollte wie es das im Kontext von Open Source tut. Wohl ist es nachvollziehbar, dass die Projektsituation in kommerziellen Software-Projekten den Programmierern weniger Möglichkeiten gibt, ihr Potential optimal in das Projekt einzubringen. Auch ist es begreiflich, wenn Projekt-Verantwortliche in Software-Firmen darauf verzichten, eine Projektvision so zu skizzieren, dass sie von den Programmierern nachvollzogen werden kann. Dies ist aber nicht zwingend. Es ist allenfalls aufwändig, eine Projektvision zu formulieren und eine optimale Herausforderung für die am Projekt beteiligten Software-Entwickler zu finden, aber es ist nicht unmöglich. Aus diesem Grund, so kann ich aus meinen Untersuchungen schliessen, ist es auch nicht unmöglich, dass ein Software-Projekt im kommerziellen Umfeld genau gleich viel Spass macht wie eines, welches unter Open-Source-Bedingungen stattfindet.

Dies postuliert stellt sich umgekehrt die Frage, ob es für einen Arbeitgeber vorteilhaft ist, die Arbeitssituation für die Software-Entwickler so zu gestalten, dass diese mit dem gleichen Spass an der Arbeit sind wie in Open-Source-Projekten. Mit anderen Worten: Tragen Programmierer, die mit Freude bei der Arbeit sind, wirklich so viel zum Unternehmenserfolg bei, dass sich der Aufwand lohnt?

Diese Frage lässt sich auf Grund zweierlei Überlegungen tendenziell positiv beantworten. Einerseits gilt generell, dass Angestellte, die Spass an der Arbeit haben, zu einem positiven Arbeitsklima beitragen, motivierter bei der Arbeit sind und weniger Fehlzeiten aufweisen. Mit den Resultaten aus Abschnitt 9.4 kann ich noch

eine spezifischere Antwort geben: Sowohl Herausforderung wie auch Projektvision, die gemäss dieser Untersuchung den Spass am Programmieren bestimmen, sind Faktoren, die vom Arbeitgeber stark beeinflusst werden können. Weiter gehe ich davon aus, dass ein Arbeitgeber einen gewissen Handlungsspielraum bei der Gestaltung von Projektvisionen oder beim Suchen von Herausforderungen für die Programmierer hat. Wenn es nun dem Arbeitgeber gelingt, eine Projektvision zu entwickeln, die nachvollziehbar und im Einklang mit dem Unternehmensziel ist, wenn er es schafft, eine Herausforderung zu bieten, welche optimal abgestimmt ist auf die Leistungsfähigkeit des Programmierers und mit der Unternehmensstrategie übereinstimmt, dann wird dies zu einem Arbeitsklima führen, in welchem die Programmierer sowohl Spass haben wie auch produktiv sind. Unter diesen Umständen wird der Spass der Software-Entwickler zweifellos positiv zum Unternehmenserfolg beitragen.

Ausblick

Während mein Versuch, den Spass am Programmieren und das Engagement in einem Open-Source-Projekt zu messen, gelungen ist und es mir ermöglicht hat, die Bedeutung von Spass zu ermitteln, bin ich mit meinem Vorhaben gescheitert, die Produktivität der Software-Entwickler zu berechnen und in eine Beziehung zum Spass zu bringen. In den Folgerungen im Kapitel 8.5 habe ich die Gründe reflektiert, warum mir dies nicht gelungen ist. In diesen Überlegungen bin ich zum Schluss gekommen, dass es nicht möglich ist, Produktivität unabhängig von Spass zu messen, weil das beste Mass für Produktivität eines Software-Entwicklers exakt dessen Spass beim Programmieren ist. Diese Behauptung wird mit dem Pugs-Projekt[1] aufs Schönste bestätigt.

Das Pugs-Projekt ist ein Open-Source-Projekt, welches innerhalb von acht Monaten eine hochproduktive Entwicklergemeinschaft von über 100 Programmierern bilden konnte (siehe Broadwell (2005)). Während die Open-Source-Projekte üblicherweise eine ausgeprägt schiefe Verteilung aufweisen, was die Anzahl Beiträge pro Beitragsleister betrifft (einige wenige Programmierer sind verantwortlich für den überwiegenden Teil der Beiträge, vgl. Spaeth (2005)), besticht das Pugs-Projekt durch eine erstaunliche Gleichverteilung der Aktivitäten. Das Geheimnis dieses Projekts: es ist „optimized for fun“. Der Projekteigentümer drückt das wie folgt aus: „The essence of fun boils down to instant gratification and a sense of wonder and discovery.“

Mit welchen Mitteln wird Spass im Pugs-Projekt optimiert?

Der erste und wichtigste Entscheid war, Spass zum primären Ziel des Projekts zu machen. Die Projektplanung und -organisaton ist in der Folge strategisch auf dieses Ziel ausgerichtet worden.

Jeder Programmierer, der am Projekt mitarbeiten will, wird sofort und umstandslos aufgenommen. Dies bedeutet konkret, dass jeder neu zum Projekt stossende Software-Entwickler postwendend das Recht bekommt, seinen Code in die Versionenverwaltung einzuchecken (das sogenannte *Committerrecht*). Der Projekteigentümer hat zu diesem Zweck eigens eine Erweiterung geschrieben, um die

[1] http://www.pugscode.org/

Vergabe des Committerrechts zu automatisieren und damit unabhängig von einer Person und ihrer Präsenz zu machen. Den beitrittswilligen Programmierern soll die Schwelle so niedrig wie möglich gemacht werden, einen Beitrag zum Projekt beizusteuern.

Die am Projekt beteiligten Personen werden bei ihrer Arbeit durch ein modernes System zur Verwaltung der Versionen unterstützt. Verglichen mit älteren Versionenverwaltungen sind die Möglichkeiten für dezentralisiertes Arbeiten offline in den modernen Systemen erheblich verbessert worden. Mit modernen Versionenverwaltungen hat jeder Projekt-Programmierer in jeder Phase seiner Arbeit die Sicherheit, dass sein Beitrag konsistent ins System übertragen wird, ohne dabei die Arbeit der anderen Projektbeteiligten zu stören oder von diesen gestört zu werden. Wird durch ein Checkin das System inkonsistent, d.h. die neuen Ergänzungen in einem Teil führen zu Fehlfunktionen in einem anderen Teil, so kann dieser Zusammenhang sehr schnell eruiert und der konsistente Systemzustand wiederhergestellt werden. Auf diese Weise bildet die Versionenverwaltung ein Sicherheitsnetz, über welchem die Software-Entwickler, ob Anfänger oder Professional, befreit und unbeschwert programmieren können.

Ein funktionierendes Programm macht mehr Spass als konzeptionelle Ideen. Der Projekteigentümer drängt die Projektbeteiligten, ihre Vorstellungen so bald wie möglich in Programmcode umzusetzen, seien diese Ideen noch so provisorisch. Sobald solcher Code vorliegt, können andere Entwickler diese Ideen aufgreifen und damit spielen. Auch zu diesem Zweck ist eine öffentlich zugängliche Versionenverwaltung unumgänglich.

Der Schlüssel des Erfolgs des Pugs-Projekts liegt demnach darin, dass die Programmierer nicht daran gehindert werden, Wirkung zu erzielen. Bei gegebener Motivation der Software-Entwickler, sonst würden sie sich ja nicht für das Projekt interessieren und sich in diesem engagieren, geht es darum, diese Motivation möglichst ohne Hindernisse in Code umzuwandeln. Jede Anmeldung, jedes Warten auf eine Antwort erzeugt Reibung und damit Unsicherheit und potentiell Frustration, jede reibungslose Interaktion dagegen Wirkung, z.B. funktionierenden Code, und damit Spass. Das Pugs-Projekt bestätigt damit meine Ableitung: Produktivität besteht darin, eine Idee verwirklichen zu können, und dies macht Spass.

In meiner Dissertation habe ich den Spass am Programmieren in den Mittelpunkt meiner Untersuchung des Open-Source-Phänomens gestellt. Mit Bestimmtheit ist dieses Phänomen zu vielfältig, als dass es sich allein auf das Element „Spass“ reduzieren liesse. Aber das Phänomen „Open Source“ bietet einen hervorragenden Rahmen, um über Spass nachzudenken, über dessen Beziehung zu Produktivität und Innovation, nicht zuletzt auch über den Stellenwert von Spass, beim Programmieren, in der Arbeit allgemein oder in der Gesellschaft überhaupt.

Anhang A

Die Open-Source-Definition

(aus Open Source Initiative (1999), Version 1.9)

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
***Rationale**: By constraining the license to require free redistribution, we eliminate the temptation to throw away many long-term gains in order to make a few short-term sales dollars. If we didn't do this, there would be lots of pressure for cooperators to defect.*

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.
***Rationale**: We require access to un-obfuscated source code because you can't evolve programs without modifying them. Since our purpose is to make evolution easy, we require that modification be made easy.*

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
***Rationale**: The mere ability to read source isn't enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.*

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of „patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
***Rationale**: Encouraging lots of improvement is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.*

Accordingly, an open-source license must guarantee that source be readily available, but may require that it be distributed as pristine base sources plus patches. In this way, „unofficial" changes can be made available but readily distinguished from the base source.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.
***Rationale**: In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore we forbid any open-source license from locking anybody out of the process.*

Some countries, including the United States, have export restrictions for certain types of software. An OSD-conformant license may warn licensees of applicable restrictions and remind them that they are obliged to obey the law; however, it may not incorporate such restrictions itself.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
***Rationale**: The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. We want commercial users to join our community, not feel excluded from it.*

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
***Rationale**: This clause is intended to forbid closing up software by indirect means such as requiring a non-disclosure agreement.*

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
***Rationale**: This clause forecloses yet another class of license traps.*

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.
***Rationale**: Distributors of open-source software have the right to make their own choices about their own software.*

Yes, the GPL is conformant with this requirement. Software linked with GPLed libraries only inherits the GPL if it forms a single work, not any software with which they are merely distributed.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.
***Rationale**: This provision is aimed specifically at licenses which require an explicit gesture of assent in order to establish a contract between licensor and licensee. Provisions mandating so-called "click-wrap"may conflict with important methods of software distribution such as FTP download, CD-ROM anthologies, and web mirroring; such provisions may also hinder code re-use. Conformant licenses must allow for the possibility that (a) redistribution of the software will take place over non-Web channels that do not support click-wrapping of the download, and that (b) the covered code (or re-used portions of covered code) may run in a non-GUI environment that cannot support popup dialogues.*

Anhang B

Herleitung der Produktionsfunktion

Ein sinnvoller Startpunkt für die Herleitung einer Produktionsfunktion zur Erklärung des freiwilligen und unbezahlten Engagements für Open Source kann in der Theorie kompensierender Lohndifferentiale (Lorenz und Wagner (1988), Backes-Gellner, Lazear und Wolff (2001)) gefunden werden. Diese Theorie besagt im Wesentlichen, dass Arbeitnehmer indifferent sind gegenüber mehr Lohn und mehr Freizeit. Eine Arbeitssituation, die durch hohe Zeitautonomie gekoppelt mit weniger Lohn (pro Zeiteinheit) charakterisiert ist, hat für die Arbeitnehmer den gleichen Nutzen wie eine Situation, in welcher sie weniger Arbeitszeitsouveränität geniessen, dafür aber mehr Lohn erhalten (Backes-Gellner u. a., 2001, S. 409 ff). Gemäss diesem Modell sind also die Arbeitnehmer bereit, zugunsten von Arbeitszeitsouveränität auf einen Anteil von Lohn zu verzichten bzw. sie müssen bei geringer Zeitflexibilität am Arbeitsplatz durch einen höheren Lohn kompensiert werden. Ökonometrische Untersuchungen zeigen, dass die Indifferenzkurven eine konvexe Form (vgl. Grafik B.1) haben.

(B.1) $$w = c - q_1 * A + q_2 * A^2$$

wo w: Lohn
A: Arbeitszeitsouveränität
$q_1, q_2 > 0$

Die Theorie kompensierender Lohndifferentiale kann auch Lohnunterschiede erklären, die durch anders geartete Variablen verursacht werden als durch Arbeitszeitsouveränität. Im Falle von Software-Entwicklung ist folgendes Modell nahe liegend: Je mehr Spass Software-Entwickler an ihrer Tätigkeit haben, umso mehr sind sie bereit, auf Lohnanteile zu verzichten. Oder umgekehrt: Müssen die Programmierer Tätigkeiten ausüben, die ihnen wenig Spass machen, so müssen sie dafür durch höheren Lohn kompensiert werden.

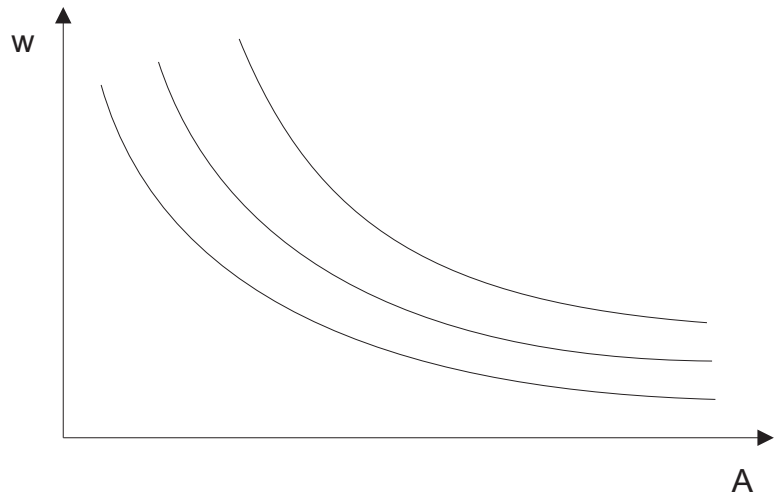


Abbildung B.1: Indifferenzkurven von Arbeitnehmern bei der Wahl zwischen Arbeitszeitsouveränität und Lohn

(B.2) $w = c_0 - a_1 * S + a_2 * S^2$

wo w: Lohn
S: Spass
$a_1, a_2 > 0$

In meiner Untersuchung interessiere ich mich aber nicht für den Lohn als abhängige Variable. Mich interessieren die Motive für das freiwillige und unentgeltliche Engagement der Software-Entwickler im Open-Source-Bereich. Zu diesem Zweck definiere ich freiwilliges Engagement E als Differenz zwischen einem Lohn w_0, den die Programmierer verlangen würden für eine Tätigkeit, die absolut keinen Spass macht, und dem effektiven Lohn:

(B.3) $E = w_0 - w$

wo E: freiwilliges, unentgeltliches Engagement
w_0: Lohn für freudlose Tätigkeit

Diese Gleichung eingesetzt in (B.2) ergibt:

(B.4) $E = c_1 + a_1 * S - a_2 * S^2$

wo $c_1 = w_0 - c_0$

Da ich $a_2 > 0$ annehme, beschreibt diese Gleichung eine konkave Funktion. Dies besagt, dass der Grenzeffekt von zusätzlichem Spass fallend ist. Dies entspricht den üblichen Annahmen in der Ökonomie.

Ich gehe davon aus, dass die Entwicklung von Open-Source-Software zu ei-

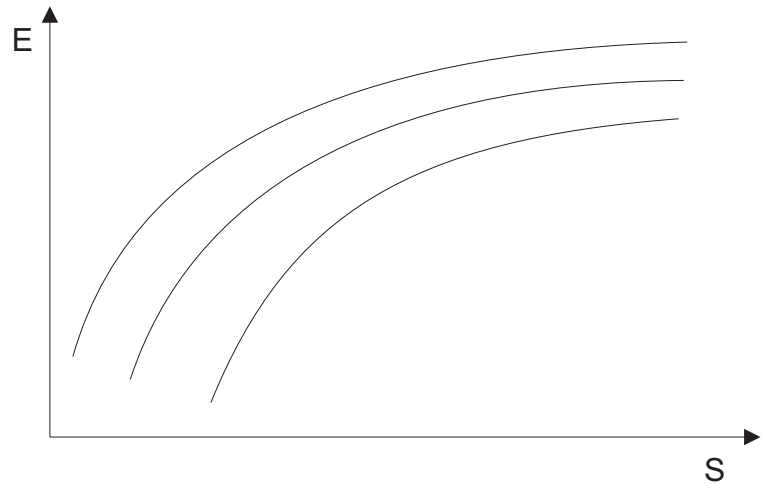


Abbildung B.2: Zusammenhang zwischen Spass und Engagement

nem grossen Teil in der Freizeit geschieht. Damit wird aber das Engagement der Entwickler nicht nur abhängig vom Spass, den sie bei der Tätigkeit empfinden, sondern auch davon, ob und wie viel Freizeit die Entwickler haben. Mit anderen Worten, die Opportunitätskosten der Zeit müssen berücksichtigt werden, um das Engagement der Open-Source-Entwickler zu erklären.

Mit einer analogen Betrachtungsweise will ich diese Opportunitätskosten der Zeit modellieren. Ich gehe davon aus, dass Arbeitnehmer indifferent sind gegenüber einer Situation mit hohem Lohn (pro Zeiteinheit) und wenig Freizeit verglichen mit einer anderen Situation, die durch weniger Lohn, dafür mehr Freizeit gekennzeichnet ist.

(B.5) $w = c_2 - b_1 * Z + b_2 * Z^2$

wo w: Lohn
Z: Freizeit
$b_1, b_2 > 0$

Wieder leite ich freiwilliges, unentgeltliches Engagement mit Hilfe der Beziehung (B.3) aus dem Lohn ab. Daraus ergibt sich:

(B.6) $E = c_3 + b_1 * Z - b_2 * Z^2$

wo $c_3 = w_0 - c_2$

Die Grafik B.3, welche die Beziehung (B.6) visualisiert, weist auf den intuitiv nachvollziehbaren Sachverhalt hin, dass umso mehr freiwilliges Engagement gezeigt wird, je mehr Freizeit einer Person zur Verfügung steht. Wieder ist mit $b_2 > 0$ die Funktion konkav, d.h. der Grenzeffekt von zusätzlicher Freizeit fällt.

Mein Modell, um das Engagement von Open-Source-Entwicklern zu erklären, sieht demnach wie folgt aus:

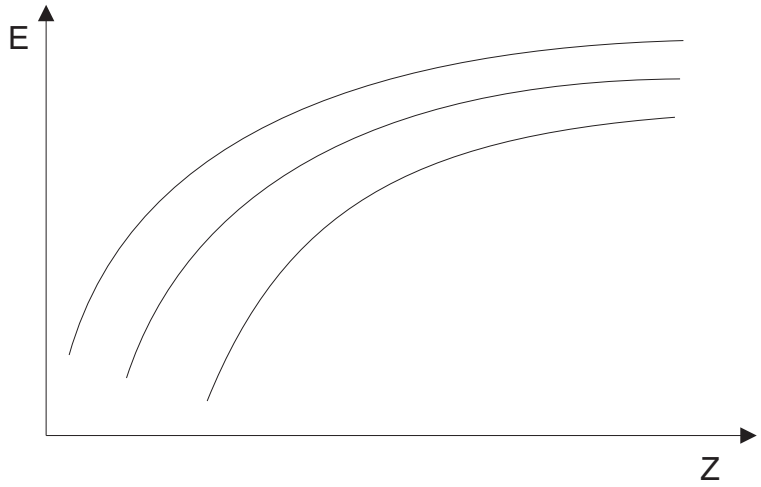


Abbildung B.3: Zusammenhang zwischen Freizeit und Engagement

(B.7) $E = c_4 + a_1 * S - a_2 * S^2 + b_1 * Z - b_2 * Z^2$

wo E: freiwilliges, unentgeltliches Engagement
S: Spass
Z: Freizeit
$a_1, a_2, b_1, b_2 > 0$

Anhang C

Der FASD-Fragebogen

Tabelle C.1: Fragebogen für Open-Source-Entwickler (englisch)

How do you feel about being an open source developer?							
How often do the following statements apply to you?		never					always
1	I lose my sense of time.	o	o	o	o	o	o
2	I cannot say how long I've been with programming.	o	o	o	o	o	o
3	I am in a state of flow when I'm working.	o	o	o	o	o	o
4	I forget all my worries when I'm working.	o	o	o	o	o	o
5	It's easy for me to concentrate.	o	o	o	o	o	o
6	I'm all wrapped up in the action.	o	o	o	o	o	o
7	I am absolutely focused on what I'm programming.	o	o	o	o	o	o
8	The requirements of my work are clear to me.	o	o	o	o	o	o
9	I hardly think of the past or the future.	o	o	o	o	o	o
10	I know exactly what is required of me.	o	o	o	o	o	o
11	There are many things I would prefer doing.	o	o	o	o	o	o
12	I feel that I can cope well with the demands of the situation.	o	o	o	o	o	o
13	My work is solely motivated by the fact that it will pay for me.	o	o	o	o	o	o
14	I always know exactly what I have to do.	o	o	o	o	o	o
15	I'm very absent-minded.	o	o	o	o	o	o
16	I don't have to muse over other things.	o	o	o	o	o	o
17	I know how to set about it.	o	o	o	o	o	o
18	I'm completely focused.	o	o	o	o	o	o
19	I feel able to handle the problem.	o	o	o	o	o	o
20	I am extremely concentrated.	o	o	o	o	o	o
21	I'm looking forward to my programming work.	o	o	o	o	o	o
22	I enjoy my work.	o	o	o	o	o	o
23	I feel the demands upon me are excessive.	o	o	o	o	o	o
24	Things just seem to fall into place.	o	o	o	o	o	o
25	I forget everything around me.	o	o	o	o	o	o
26	I accomplish my work for its own sake.	o	o	o	o	o	o

27	I completely concentrate on my programming work.	o	o	o	o	o	o
28	I am easily distracted by other things.	o	o	o	o	o	o
		applies not at all					applies completely
29	I'm looking forward to further development activities for open source software.	o	o	o	o	o	o
30	I'm prepared to increase my future commitment in the development of open source software.	o	o	o	o	o	o
31	With one more hour in the day, I would program open source software.	o	o	o	o	o	o
	How do you feel about the open source projects?	never					always
32	How often are the open source projects you work on based on a definite project vision?	o	o	o	o	o	o
33	How often is there a deadline for your open source projects?	o	o	o	o	o	o
		absolutely irrelevant					very important
34	How important is the vision behind an open source project for you to participate in the project?	o	o	o	o	o	o
35	How important is the professional competence of the project leader for your commitment in an open source project?	o	o	o	o	o	o
36	As a project member: What are your reasons for participating in open source projects?						
a	It was important for my work to have a certain functionality; that's why I joined the open source project and got involved.	o	o	o	o	o	o
b	My employer asked me to participate in the open source project because he needed its functionality.	o	o	o	o	o	o
c	Because I wanted to do something for the open source community.	o	o	o	o	o	o
d	The project promised to be fun.	o	o	o	o	o	o
e	My colleagues motivated me to participate in the open source project.	o	o	o	o	o	o
f	By participating in the open source project you could become famous.	o	o	o	o	o	o
g	Because I wanted to learn and develop new skills.	o	o	o	o	o	o
37	As a project leader: For what reasons did you initiate your open source project(s)?						
a	I needed a certain functionality for my work, and I wanted to make this functionality available as an open source application.	o	o	o	o	o	o

b	My employer asked me to start the open source project because he needed the functionality.	o	o	o	o	o	o
c	My employer asked me to start the open source project because he could earn money with the application.	o	o	o	o	o	o
d	Because I wanted to do something for the open source community.	o	o	o	o	o	o
e	One open source project in the past didn't develop as desired, therefore I had to start my own.	o	o	o	o	o	o
f	The project promised to be fun.	o	o	o	o	o	o
g	I needed assistants to complete a software project.	o	o	o	o	o	o
h	With an open source project you could become famous.	o	o	o	o	o	o

How do you organize your time and commitment in the open source area?

38	How many patches have you developed for open source software?	Auswahl [1]	
39	How many classes/modules/files etc. have you developed for open source software?	Auswahl	
40	Please estimate the time you spend for the development of open source software (average hours per week).	Eingabe	hours
41	Of the total time spent for the development of open source software, how much in percent is part of your spare time?	Auswahl [2]	%
42	How much (on average, in percent) of your spare time do you spend on activities concerning open source projects?	Auswahl	%
43	Please specify the most important position you hold or held in an open source project.	Auswahl [3]	

Demographic data:

44	Please state the year in which you started to develop open source software.	Auswahl [4]	
45	How old were you when you started to develop open source software?	Auswahl [5]	
46	Do you have children?	Auswahl [6]	
47	Are there other adults living in the same household with you?	Auswahl	
48	How many pastimes (including programming) do you have?	Auswahl [7]	
49	Gender	Auswahl [8]	
50	Nationality	Auswahl [9]	

[1] Auswahlmenge: 0, 1-3, 4-10, 11-50, 51-100, 101-200, 201-500, 501-1'000, 1'001-2'000, 2'001-5'000, 5'001-10'000, more then 10'000.

[2] Auswahlmenge: 0-10, 11-20, 21-30, 31-40, 41-50, 51-60, 61-70, 71-80, 81-90, 91-100.

[3] Auswahlmenge: project leader, developer, bug fixer, subscribed to a mailing list, user.

[4] Auswahlmenge: Jahr 1961 - 2004.

[5] Auswahlmenge: Alter 10 - 60.

[6] Auswahlmenge: yes, no.

[7] Auswahlmenge: 1, 2, 3, 4, 5, more then 5.

[8] Auswahlmenge: male, female.

[9] Auswahlmenge: alle Namen von Staaten.

51	Occupation	Auswahl [10]	
And finally:			
52	How much time did it take you to answer this questionnaire?	Eingabe	Minutes
53	If you have questions or remarks concerning this questionnaire, please enter them in this field:	Eingabe	

[10] Auswahlmenge: 100% employed, 90% - 99% employed, 80% - 89% employed, 70% - 79% employed, 60% - 69% employed, 50% - 59% employed, 40% - 49% employed, 30% - 39% employed, 20% - 29% employed, 10% - 19% employed, student, unemployed.

Tabelle C.2: Fragebogen für Open-Source-Entwickler (deutsch)

Wie erleben Sie Ihre Tätigkeit als Open Source Programmierer?							
	Wie oft treffen die folgenden Aussagen auf Sie zu?	nie					immer
1	Ich vergesse ganz die Zeit.	o	o	o	o	o	o
2	Ich habe kein Gefühl mehr dafür, wie lange ich schon am Programmieren bin.	o	o	o	o	o	o
3	Die Handlung erfolgt wie aus einem Guss.	o	o	o	o	o	o
4	Ich vergesse meine Sorgen.	o	o	o	o	o	o
5	Es fällt mir leicht, mich zu konzentrieren.	o	o	o	o	o	o
6	Ich gehe ganz in der Handlung auf.	o	o	o	o	o	o
7	Meine Aufmerksamkeit ist völlig aufs Programmieren gelenkt.	o	o	o	o	o	o
8	In der Situation sind die Anforderungen an mich völlig klar.	o	o	o	o	o	o
9	An Vergangenes oder Zukünftiges denke ich kaum.	o	o	o	o	o	o
10	Die Forderungen an mich sind eindeutig.	o	o	o	o	o	o
11	Es gibt viele Dinge, die ich lieber tun würde.	o	o	o	o	o	o
12	Ich habe das Gefühl, die Anforderungen der Situation gut bewältigen zu können.	o	o	o	o	o	o
13	Ich handle nur, weil ich weiss, dass es sich für mich bezahlt machen wird.	o	o	o	o	o	o
14	Ich weiss immer genau, was zu tun ist.	o	o	o	o	o	o
15	Ich bin mit meinen Gedanken ganz woanders.	o	o	o	o	o	o
16	Ich muss nicht über andere Dinge grübeln.	o	o	o	o	o	o
17	Mir ist klar, wie ich vorzugehen habe.	o	o	o	o	o	o
18	Ich bin voll und ganz bei der Sache.	o	o	o	o	o	o
19	Ich fühle mich der Aufgabe gewachsen.	o	o	o	o	o	o
20	Meine Konzentration ist sehr hoch.	o	o	o	o	o	o
21	Ich freue mich schon vorher aufs Programmieren.	o	o	o	o	o	o
22	Die Handlung macht mir Spass.	o	o	o	o	o	o
23	Ich fühle mich überfordert.	o	o	o	o	o	o
24	Alles scheint wie von selbst zu laufen.	o	o	o	o	o	o
25	Ich vergesse alles um mich herum.	o	o	o	o	o	o
26	Ich führe die Handlung um ihrer selbst willen aus.	o	o	o	o	o	o
27	Ich konzentriere mich voll aufs Programmieren.	o	o	o	o	o	o
28	Ich lasse mich von anderen Dingen ablenken.	o	o	o	o	o	o
		trifft keinesfalls zu					trifft völlig zu
29	Ich freue mich auf weitere Entwicklungstätigkeiten für Open Source Software.	o	o	o	o	o	o

30	Ich bin bereit, mein zeitliches Engagement bei der Entwicklung von Open Source Software in Zukunft zu vergrössern.	o	o	o	o	o	o
31	Eine zusätzliche Stunde Zeit im Tag würde ich zum Programmieren von Open Source verwenden.	o	o	o	o	o	o

Wie erleben Sie die Open Source Projekte?		nie					immer
32	Wie oft sind die Open Source Projekte, an denen Sie sich beteiligen, von einer klaren Projektvision getragen?	o	o	o	o	o	o
33	Bei Ihrer Arbeit für Open Source Projekte: Wie häufig sind Abgabetermine spürbar?	o	o	o	o	o	o

		völlig unwichtig				sehr wichtig
34	Wie wichtig ist die Vision hinter einem Open Source Projekt für Ihre Projektbeteiligung?	o	o	o	o	o	o
35	Wie wichtig ist die fachliche Kompetenz des Projekteigentümers für Ihr Engagement an einem Open Source Projekt?	o	o	o	o	o	o
36	Als Projektmitglied: Aus welchen Gründen beteiligten Sie sich an Open Source Projekten?						
a	Ich benötigte (für meine Arbeit) eine bestimmte Funktionalität und bin so auf das Open Source Projekt gestossen und habe mich darin engagiert.	o	o	o	o	o	o
b	Mein Arbeitgeber beauftragte mich, am Open Source Projekt mitzuarbeiten, weil er die Funktionalität benötigte.	o	o	o	o	o	o
c	Weil ich etwas für die Open Source Community machen wollte.	o	o	o	o	o	o
d	Das Projekt versprach, Spass zu machen.	o	o	o	o	o	o
e	Kollegen haben mich dazu motiviert, am Open Source Projekt mitzuarbeiten.	o	o	o	o	o	o
f	Durch die Beteiligung am Open Source Projekt konnte man berühmt werden.	o	o	o	o	o	o
g	Weil ich neue Fähigkeiten erwerben und entwickeln wollte.	o	o	o	o	o	o
37	Als Projektverantwortlicher: Aus welchen Gründen initiierten Sie Ihre Open Source Projekte?						
a	Ich benötigte (für meine Arbeit) eine bestimmte Funktionalität und wollte diese in der Form einer Open Source Applikation zur Verfügung stellen.	o	o	o	o	o	o
b	Mein Arbeitgeber beauftragte mich, das Open Source Projekt zu starten, weil er die Funktionalität benötigte.	o	o	o	o	o	o
c	Mein Arbeitgeber beauftragte mich, das Open Source Projekt zu starten, weil er mit der Applikation Geld verdienen konnte.	o	o	o	o	o	o
d	Weil ich etwas für die Open Source Community machen wollte.	o	o	o	o	o	o

e	Ein älteres Open Source Projekt hat sich nicht in gewünschter Weise entwickelt und so musste ich ein eigenes starten.	o	o	o	o	o	o
f	Das Projekt versprach, Spass zu machen.	o	o	o	o	o	o
g	Ich benötigte Helfer, um ein Software-Projekt fertigstellen zu können.	o	o	o	o	o	o
h	Mit einem Open Source Projekt konnte man berühmt werden.	o	o	o	o	o	o

Wie organisieren Sie Ihr Engagement im Open Source Bereich?

38	Wie viele Patches haben Sie für Open Source Software geschrieben?	Auswahl [11]	
39	Wie viele Klassen/Module/Files etc. haben Sie für Open Source Software geschrieben?	Auswahl	
40	Bitte schätzen Sie die Zeit, die sie aktuell für die Entwicklung von Open Source Software aufwenden (durchschnittliche Stunden pro Woche).	Eingabe	Stunden
41	Von der Zeit insgesamt, die Sie für die Entwicklung von Open Source Software verwenden, wie viel Prozent entwickeln Sie in Ihrer Freizeit?	Auswahl [12]	%
42	Wie viel Prozent Ihrer Freizeit verwenden Sie für Arbeiten in Open Source Projekten (im Durchschnitt)?	Auswahl	%
43	Bitte geben Sie die wichtigste Position an, die Sie in einem der Open Source Projekte einnehmen bzw. eingenommen haben.	Auswahl [13]	

Statistische Angaben:

44	In welchem Jahr haben Sie angefangen, Open Source Software zu entwickeln?	Auswahl [14]	
45	Wie alt waren Sie, als sie angefangen haben, Open Source Software zu entwickeln?	Auswahl [15]	
46	Haben Sie Kinder?	Auswahl [16]	
47	Leben in Ihrem Haushalt weitere erwachsene Personen?	Auswahl	
48	Mit wie vielen Hobbies (inklusive Programmieren) beschäftigen Sie sich regelmässig?	Auswahl [17]	
49	Geschlecht	Auswahl [18]	
50	Nationalität	Auswahl [19]	

[11] Auswahlmenge: 0, 1-3, 4-10, 11-50, 51-100, 101-200, 201-500, 501-1'000, 1'001-2'000, 2'001-5'000, 5'001-10'000, mehr als 10'000.

[12] Auswahlmenge: 0-10, 11-20, 21-30, 31-40, 41-50, 51-60, 61-70, 71-80, 81-90, 91-100.

[13] Auswahlmenge: Projektverantwortlicher, Entwickler, Bug-Fixer, eingeschrieben in Mailing-Liste, Benutzer.

[14] Auswahlmenge: Jahr 1961 - 2004.

[15] Auswahlmenge: Alter 10 - 60.

[16] Auswahlmenge: ja, nein

[17] Auswahlmenge: 1, 2, 3, 4, 5, mehr als 5.

[18] Auswahlmenge: männlich, weiblich.

[19] Auswahlmenge: alle Namen von Staaten.

51	Beschäftigung	Auswahl [20]	
Zum Abschluss:			
52	Wie viel Zeit benötigten Sie für die Beantwortung dieses Fragebogens?	Eingabe	Minuten
53	Falls Sie Fragen und Bemerkungen zum Fragebogen und zur Umfrage haben, geben Sie diese bitte in diesem Feld ein:	Eingabe	

[20] Auswahlmenge: 100% beschäftigt, 90% - 99% beschäftigt, 80% - 89% beschäftigt, 70% - 79% beschäftigt, 60% - 69% beschäftigt, 50% - 59% beschäftigt, 40% - 49% beschäftigt, 30% - 39% beschäftigt, 20% - 29% beschäftigt, 10% - 19% beschäftigt, Student, arbeitslos.

Tabelle C.3: Fragebogen für kommerzielle Software-Entwickler (englisch)

How do you feel about being a developer?							
How often do the following statements apply to you?		never					always
1	I lose my sense of time.	o	o	o	o	o	o
2	I cannot say how long I've been with programming.	o	o	o	o	o	o
3	I am in a state of flow when I'm working.	o	o	o	o	o	o
4	I forget all my worries when I'm working.	o	o	o	o	o	o
5	It's easy for me to concentrate.	o	o	o	o	o	o
6	I'm all wrapped up in the action.	o	o	o	o	o	o
7	I am absolutely focused on what I'm programming.	o	o	o	o	o	o
8	The requirements of my work are clear to me.	o	o	o	o	o	o
9	I hardly think of the past or the future.	o	o	o	o	o	o
10	I know exactly what is required of me.	o	o	o	o	o	o
11	There are many things I would prefer doing.	o	o	o	o	o	o
12	I feel that I can cope well with the demands of the situation.	o	o	o	o	o	o
13	My work is solely motivated by the fact that it will pay for me.	o	o	o	o	o	o
14	I always know exactly what I have to do.	o	o	o	o	o	o
15	I'm very absent-minded.	o	o	o	o	o	o
16	I don't have to muse over other things.	o	o	o	o	o	o
17	I know how to set about it.	o	o	o	o	o	o
18	I'm completely focused.	o	o	o	o	o	o
19	I feel able to handle the problem.	o	o	o	o	o	o
20	I am extremely concentrated.	o	o	o	o	o	o
21	I'm looking forward to my programming work.	o	o	o	o	o	o
22	I enjoy my work.	o	o	o	o	o	o
23	I feel the demands upon me are excessive.	o	o	o	o	o	o
24	Things just seem to fall into place.	o	o	o	o	o	o
25	I forget everything around me.	o	o	o	o	o	o
26	I accomplish my work for its own sake.	o	o	o	o	o	o
27	I completely concentrate on my programming work.	o	o	o	o	o	o
28	I am easily distracted by other things.	o	o	o	o	o	o
Workplace and employer		never					always
29	At the beginning of the week, I look forward to the work ahead.	o	o	o	o	o	o
30	I don't mind working overtime for my job.	o	o	o	o	o	o
31	I plan my future career in the IT area.	o	o	o	o	o	o
32	I'm proud to work for the *firm*[21].	o	o	o	o	o	o
33	I can develop personally and professionally working for the *firm*.	o	o	o	o	o	o

[21] Im Online-Fragebogen wurde der Name der Firma angezeigt.

34	There is a pleasant atmosphere at the *firm*, and the firm pursues the right objectives.	o	o	o	o	o	o
35	At the *firm*, the role model and the processes are implemented professionally and with success.	o	o	o	o	o	o
Connection to Open Source		never					always
36	In my working hours, I work on Open Source projects.	o	o	o	o	o	o
37	At my workplace, we use Open Source software.	o	o	o	o	o	o
What is your experience with this project work?		nie					immer
38	At your workplace: How often do you feel deadlines?	o	o	o	o	o	o
39	At your workplace: How often are the projects supported by a clear project vision?	o	o	o	o	o	o
40	At your workplace: How often do you feel the formal authority of the project manager?	o	o	o	o	o	o
		absolutely irrelevant				very	important
41	At your workplace: How important is a vision supporting the software project for your work as software developer?	o	o	o	o	o	o
42	At your workplace: How important is the project manager's professional competence for your work in a software project?	o	o	o	o	o	o
43	How much of your time at your workplace are you acting as a *project manager*?	Auswahl [22]					
44	How much of your time at your workplace are you acting as a *software architect*?	Auswahl					
45	How much of your time at your workplace are you acting as a *software developer*? *Note: The statements don't have to sum up to 100%.*	Auswahl					
46	How many check-ins did you do on your last workday?	Eingabe			Checkins		
47	How many check-ins were you able to do in the past five workdays?	Eingabe			Checkins		
Demographic data:							
48	Please state the year in which you started to develop software.	Auswahl [23]					
49	How old were you when you started to develop software?	Auswahl [24]					
50	Occupation	Auswahl [25]					

[22] Auswahlmenge: 91% - 100%, 81% - 90%, 71% - 80%, 61% - 70%, 51% - 60%, 41% - 50%, 31% - 40%, 21% - 30%, 11% - 20%, 0% - 10% .

[23] Auswahlmenge: Jahr 1961 - 2004.

[24] Auswahlmenge: Alter 10 - 60.

[25] Auswahlmenge: 100% employed, 90% - 99% employed, 80% - 89% employed, 70% - 79% employed, 60% - 69% employed, 50% - 59% employed, 40% - 49% employed, 30% - 39% employed,

51	Gender	Auswahl [26]	
Zum Abschluss:			
And finally:			
52	How much time did it take you to answer this questionnaire?	Eingabe	Minutes
53	If you have questions or remarks concerning this questionnaire, please enter them in this field:	Eingabe	

20% - 29% employed, 10% - 19% employed.

[26] Auswahlmenge: male, female.

Tabelle C.4: Fragebogen für kommerzielle Software-Entwickler (deutsch)

Wie erleben Sie Ihre Tätigkeit als Programmierer?							
Wie oft treffen die folgenden Aussagen auf Sie zu?		nie					immer
1	Ich vergesse ganz die Zeit.	o	o	o	o	o	o
2	Ich habe kein Gefühl mehr dafür, wie lange ich schon am Programmieren bin.	o	o	o	o	o	o
3	Die Handlung erfolgt wie aus einem Guss.	o	o	o	o	o	o
4	Ich vergesse meine Sorgen.	o	o	o	o	o	o
5	Es fällt mir leicht, mich zu konzentrieren.	o	o	o	o	o	o
6	Ich gehe ganz in der Handlung auf.	o	o	o	o	o	o
7	Meine Aufmerksamkeit ist völlig aufs Programmieren gelenkt.	o	o	o	o	o	o
8	In der Situation sind die Anforderungen an mich völlig klar.	o	o	o	o	o	o
9	An Vergangenes oder Zukünftiges denke ich kaum.	o	o	o	o	o	o
10	Die Forderungen an mich sind eindeutig.	o	o	o	o	o	o
11	Es gibt viele Dinge, die ich lieber tun würde.	o	o	o	o	o	o
12	Ich habe das Gefühl, die Anforderungen der Situation gut bewältigen zu können.	o	o	o	o	o	o
13	Ich handle nur, weil ich weiss, dass es sich für mich bezahlt machen wird.	o	o	o	o	o	o
14	Ich weiss immer genau, was zu tun ist.	o	o	o	o	o	o
15	Ich bin mit meinen Gedanken ganz woanders.	o	o	o	o	o	o
16	Ich muss nicht über andere Dinge grübeln.	o	o	o	o	o	o
17	Mir ist klar, wie ich vorzugehen habe.	o	o	o	o	o	o
18	Ich bin voll und ganz bei der Sache.	o	o	o	o	o	o
19	Ich fühle mich der Aufgabe gewachsen.	o	o	o	o	o	o
20	Meine Konzentration ist sehr hoch.	o	o	o	o	o	o
21	Ich freue mich schon vorher aufs Programmieren.	o	o	o	o	o	o
22	Die Handlung macht mir Spass.	o	o	o	o	o	o
23	Ich fühle mich überfordert.	o	o	o	o	o	o
24	Alles scheint wie von selbst zu laufen.	o	o	o	o	o	o
25	Ich vergesse alles um mich herum.	o	o	o	o	o	o
26	Ich führe die Handlung um ihrer selbst willen aus.	o	o	o	o	o	o
27	Ich konzentriere mich voll aufs Programmieren.	o	o	o	o	o	o
28	Ich lasse mich von anderen Dingen ablenken.	o	o	o	o	o	o
Arbeitsplatz und Arbeitgeber		nie					immer
29	Am Wochenanfang freue ich mich auf die bevorstehende Arbeit.	o	o	o	o	o	o
30	Für meine Arbeit Überstunden zu machen macht mir nichts aus.	o	o	o	o	o	o
31	Meine zukünftige Karriere plane ich im IT-Bereich.	o	o	o	o	o	o

32	Ich bin stolz darauf, bei der *Firma*[27] zu arbeiten.	o	o	o	o	o	o
33	Ich kann mich bei der *Firma* persönlich und beruflich weiterentwickeln.	o	o	o	o	o	o
34	Die *Firma* ist richtig aufgestellt und verfolgt die richtigen Ziele.	o	o	o	o	o	o
35	Bei der *Firma* werden das Rollenmodell und die Prozesse professionell und erfolgreich umgesetzt.	o	o	o	o	o	o
	Beziehung zu Open Source	nie					immer
36	In meiner Arbeitszeit arbeite ich an Open-Source-Projekten.	o	o	o	o	o	o
37	An meinem Arbeitsplatz verwende ich Open-Source-Software.	o	o	o	o	o	o
	Wie erleben Sie die Projektarbeit?	nie					immer
38	An Ihrem Arbeitsplatz: Wie häufig sind Abgabetermine spürbar?	o	o	o	o	o	o
39	An Ihrem Arbeitsplatz: Wie oft sind die Software-Projekte von einer klaren Projektvision getragen?	o	o	o	o	o	o
40	An Ihrem Arbeitsplatz: Wie häufig ist die formale Autorität des Projektmanagers spürbar?	o	o	o	o	o	o
		völlig unwichtig					sehr wichtig
41	An Ihrem Arbeitsplatz: Wie wichtig ist eine Vision hinter einem Software-Projekt für ihre Arbeit als Software-Entwickler?	o	o	o	o	o	o
42	An Ihrem Arbeitsplatz: Wie wichtig ist die fachliche Kompetenz des Projektmanagers für Ihre Arbeit in einem Software-Projekt?	o	o	o	o	o	o
43	Wieviel Zeit an Ihrem Arbeitsplatz sind Sie als *Projektleiter* tätig?	Auswahl[28]					
44	Wieviel Zeit an Ihrem Arbeitsplatz sind Sie als *Software-Architekt* tätig?	Auswahl					
45	Wieviel Zeit an Ihrem Arbeitsplatz sind Sie als *Software-Entwickler* tätig? *Bemerkung: Angaben müssen sich nicht auf 100% summieren.*	Auswahl					
46	Wieviele Checkins haben Sie am letzten Arbeitstag gemacht?	Eingabe			Checkins		

[27] Im Online-Fragebogen wurde der Name der Firma angezeigt.

[28] Auswahlmenge: 91% - 100%, 81% - 90%, 71% - 80%, 61% - 70%, 51% - 60%, 41% - 50%, 31% - 40%, 21% - 30%, 11% - 20%, 0% - 10% .

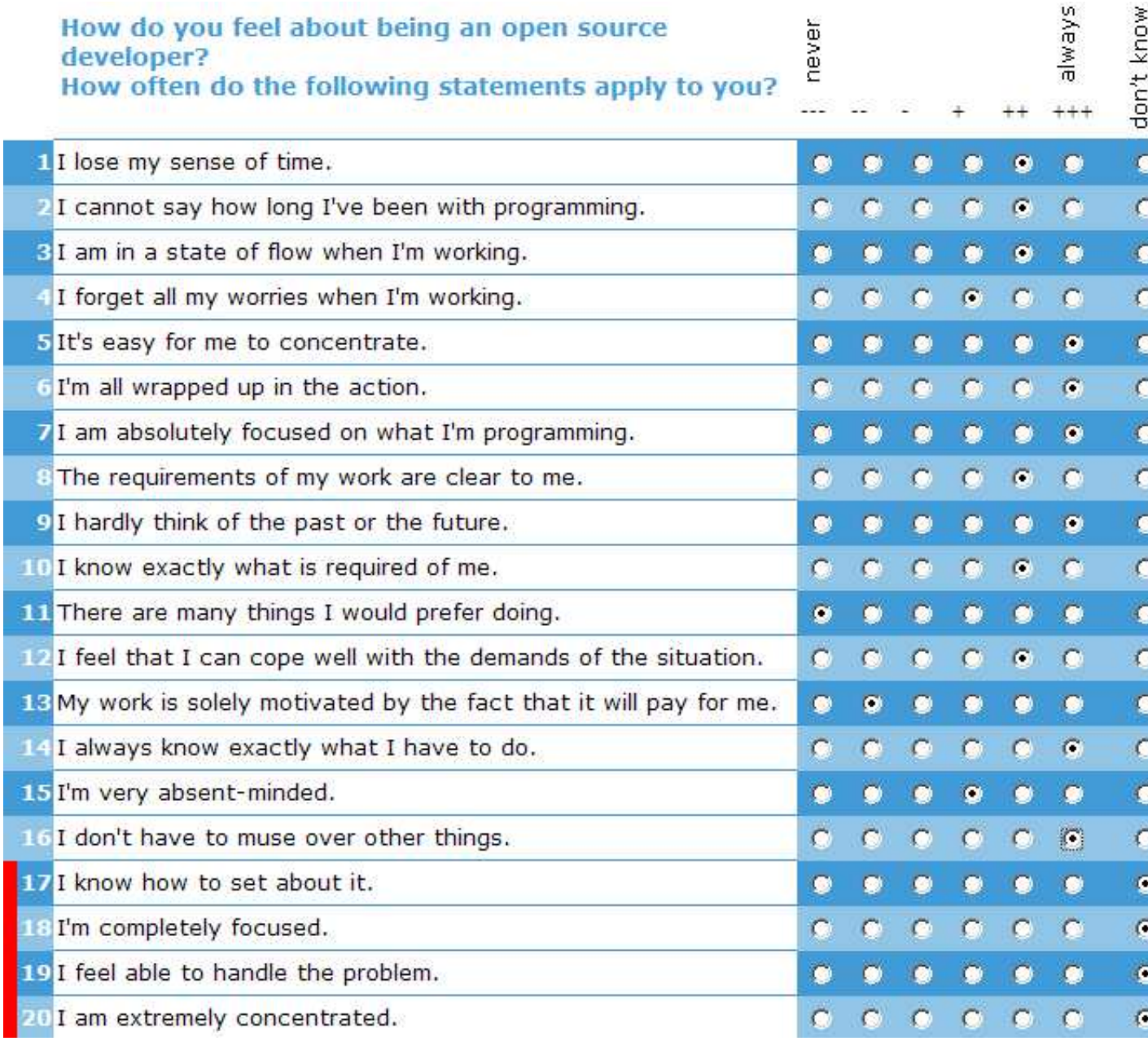
47	Wieviele Checkins haben Sie an den letzten fünf Arbeitstagen machen können?	Eingabe	Checkins
Statistische Angaben:			
48	In welchem Jahr haben Sie angefangen zu programmieren?	Auswahl [29]	
49	Wie alt waren Sie, als Sie angefangen haben zu programmieren?	Auswahl [30]	
50	Beschäftigungsgrad	Auswahl [31]	
51	Geschlecht	Auswahl [32]	
Zum Abschluss:			
52	Wie viel Zeit benötigten Sie für die Beantwortung dieses Fragebogens?	Eingabe	Minuten
53	Falls Sie Fragen und Bemerkungen zum Fragebogen und zur Umfrage haben, geben Sie diese bitte in diesem Feld ein:	Eingabe	

[29] Auswahlmenge: Jahr 1961 - 2004.

[30] Auswahlmenge: Alter 10 - 60.

[31] Auswahlmenge: 100% beschäftigt, 90% - 99% beschäftigt, 80% - 89% beschäftigt, 70% - 79% beschäftigt, 60% - 69% beschäftigt, 50% - 59% beschäftigt, 40% - 49% beschäftigt, 30% - 39% beschäftigt, 20% - 29% beschäftigt, 10% - 19% beschäftigt.

[32] Auswahlmenge: männlich, weiblich.



How do you feel about being an open source developer?
How often do the following statements apply to you?

		never ---	--	-	+	++	always +++	don't know
1	I lose my sense of time.					●		
2	I cannot say how long I've been with programming.					●		
3	I am in a state of flow when I'm working.					●		
4	I forget all my worries when I'm working.				●			
5	It's easy for me to concentrate.						●	
6	I'm all wrapped up in the action.						●	
7	I am absolutely focused on what I'm programming.						●	
8	The requirements of my work are clear to me.					●		
9	I hardly think of the past or the future.						●	
10	I know exactly what is required of me.					●		
11	There are many things I would prefer doing.	●						
12	I feel that I can cope well with the demands of the situation.					●		
13	My work is solely motivated by the fact that it will pay for me.		●					
14	I always know exactly what I have to do.						●	
15	I'm very absent-minded.				●			
16	I don't have to muse over other things.						◉	
17	I know how to set about it.							
18	I'm completely focused.							
19	I feel able to handle the problem.							
20	I am extremely concentrated.							

Abbildung C.1: Screenshot des FASD-Fragebogens

Anhang D

Lancierung der FASD-Studie

Text der Mail, mit welcher die Open-Source-Entwickler auf den Plattformen SourceForge, GNU\Savannah und BerliOS auf die FASD-Studie aufmerksam gemacht wurden:

```
Subject: FASD project: Online survey launched

Dear Open Source developer

I am doing a research project on 'Fun and Software Deve-
lopment' in which I kindly invite you to participate. You
will find the online survey under http://fasd.ethz.ch/qsf/.
The questionnaire consists of 53 questions and you will
need about 15 minutes to complete it.
   With the FASD project (Fun and Software Development) we
want to define the motivational significance of fun when
software developers decide to engage in Open Source projects.
What is special about our research project is that a similar
survey is planned with software developers in commercial
firms. This procedure allows the immediate comparison between
the involved individuals and the conditions of production
of these two development models. Thus we hope to obtain
substantial new insights to the phenomenon of Open Source
Development.

With many thanks for your participation,
Benno Luthiger

   PS: The results of the survey will be published under
http://www.isu.unizh.ch/fuehrung/blprojects/FASD/. We have
set up the mailing list fasd@webboard.ethz.ch for this study.
Please see http://fasd.ethz.ch/qsf/mailinglist_de.html for
registration to this mailing list.
```

Die in den Mails enthaltenen URLs auf die Einstiegsseite des Online-Fragebogens waren je nach Plattform unterschiedlich. Die Open-Source-Entwickler auf

SourceForge erhielten die URL *http://fasd.ethz.ch/qsf/*, die Entwickler auf GNU\
Savannah die URL *http://fasd.ethz.ch/qgs/*, während sich die Programmierer auf
BerliOS mit *http://fasd.ethz.ch/qbo/* einwählen konnten.

Antwortmail von Richard Stallman:

```
From: Richard Stallman [mailto:rms@gnu.org]
Sent: Dienstag, 4. Mai 2004 00:21
To: benno.luthiger@id.ethz.ch
Subject: Re: FASD project: Online survey launched

Thank you, but I do not consider myself an 'open source
developer', and I don't like my work to be described as
'open source'.

My work is free software (freie Software, logiciel libre).


For an explanation of the difference, please see
http://www.gnu.org/philosophy/free-software-for-freedom.html.
```

Mail zum Start der FASD-Umfrage an die Software-Entwickler einer Schweizer
Software-Firma:

```
Subject: FASD-Studie: Fun and Software Development

Lieber Software-Entwickler der Firmenname

Im Rahmen eines betriebswirtschaftlichen Forschungsprojekts
an der Universität Zürich führe ich eine Umfrage zum Thema
'Spass und Software-Entwicklung' durch. Ich bitte Sie, an
dieser Studie teilzunehmen. Die Umfrage ist unter
http://fasd.ethz.ch/qcom_AAAAAAAAAA/ abrufbar. Sie besteht
aus 53 Fragen, die Beantwortung dauert ca. 15 Min.
   Mit der FASD-Studie (Fun and Software Development) versuchen
wir herauszufinden, wie gross die Bedeutung von Spass für
die Tätigkeit der Programmierer ist. Das Spezielle an unserer
Studie ist, dass eine analoge Umfrage auch unter
Software-Entwicklern im Open-Source-Bereich durchgeführt
wird. Diese Vorgehensweise ermöglicht einen direkten Vergleich
der Motivation der Beteiligten und der Produktionsbedingungen
dieser zwei Software-Entwicklungsmodelle.
   Ihre Teilnahme an der Umfrage erfolgt anonym. Die Daten
der Untersuchung werden vertraulich behandelt.

Herzlichen Dank für Ihre Teilnahme
Benno Luthiger

   PS: Die Resultate der Umfrage werden unter
http://www.isu.unizh.ch/fuehrung/blprojects/FASD/ publiziert.
```

Für die Studie wurde die Mailingliste fasd@webboard.ethz.ch eingerichtet. Hinweise zur Anmeldung für die Mailingliste sind unter http://fasd.ethz.ch/qcom_AAAAAAAAAA/mailinglist_de.html erhältlich.

Mailvorlage für Nachfassaktion:

Anrede

An der FASD-Umfrage haben schon x% unserer Software-Entwickler teilgenommen. Ich danke/Wir danken allen Teilnehmenden für ihre Beteiligung. Damit die Ergebnisse der FASD-Studie auch für die *Firmenname* interessant wird, ist eine möglichst hohe Rücklaufquote wichtig. Die Umfrage ist noch bis zum *Datum* unter http://fasd.ethz.ch/qcom_AAAAAAAAAA/ offen. Der Zeitaufwand beträgt ca. 15 Min. Ich bitte/Wir bitten alle Software-Entwickler, die noch keine Zeit gefunden haben, den Fragebogen auszufüllen, dies noch zu tun.

Weitere Informationen zur FASD-Studie sind unter http://www.isu.unizh.ch/fuehrung/blprojects/FASD/ erhältlich.

Anhang E

Berechnung der Zeit-bezogenen Variablen

Folgende Zeit-bezogenen Variablen wurden für die statistische Auswertung berechnet:

osis: Anzahl Wochenstunden für Open Source während der Freizeit.

osaw: Anzahl Wochenstunden für Open Source während der Arbeitszeit.

sparePW: Anzahl Wochenstunden Freizeit.

workPW: Arbeitszeit Total (d.h. bezahlte und unbezahlte) in Stunden pro Woche.

pworkPW: Arbeitsstunden pro Woche bezahlt.

Zur Berechnung dieser Variablen wurden folgende Frage-Items ausgewertet:

q40: Bitte schätzen Sie die Zeit, die Sie aktuell für die Entwicklung von Open Source Software aufwenden (durchschnittliche Stunden pro Woche). *Eingabe: Anzahl Stunden*

q41: Von der Zeit insgesamt, die Sie für die Entwicklung von Open Source Software verwenden, wie viel Prozent entwickeln Sie in Ihrer Freizeit? *Eingabe: Auswahl von prozentualem Anteil in 10er Schritten von „0%-10%“ bis „91%-100%“.*

q42: Wie viel Prozent Ihrer Freizeit verwenden Sie für Arbeiten in Open Source Projekten (im Durchschnitt)? *Eingabe: Auswahl von prozentualem Anteil in 10er Schritten von „0%-10%“ bis „91%-100%“.*

q51: Beschäftigung *Auswahl von Beschäftigungskategorie aus „100% beschäftigt“, „90% - 99% beschäftigt“, „80% - 89% beschäftigt“, „70% - 79% beschäftigt“, „60% - 69% beschäftigt“, „50% - 59% beschäftigt“, „40% - 49% beschäftigt“, „30% - 39% beschäftigt“, „20% - 29% beschäftigt“, „10% - 19% beschäftigt“, „Student“, „arbeitslos“.*

Die Zeit-bezogenen Variablen wurden wie folgt berechnet:

osis $= q40 * q41/100$

osaw $= q40-$ osis

sparePW $= q40 * q41/q42$

workPW $= 112-$ sparePW

pworkPW: Aus q51 berechnet, wobei Kategorie 1 („100% beschäftigt“) auf 40 Stunden gesetzt wurde und jede nächste Kategorie um 4 Stunden weniger, die Kategorie „Student“ auf 20 Stunden und die Kategorie „arbeitslos“ auf 0 Stunden.

Die effektive Arbeitszeit pro Woche (*workPW*) kann auch mit der Methode von Bittman und Goodin (1998) berechnet werden. Im Falle meiner Arbeit zeigte sich allerdings, dass diese Methode zu wesentlich weniger gut nachvollziehbaren Resultaten führt. Insbesondere konnte ich auch keinerlei Korrelation finden mit den Werten, die ich nach der oben geschilderten Weise berechnet hatte. Aus diesem Grund wurde die Methode von Bittman und Goodin nicht weiter verfolgt.

Literaturverzeichnis

[Amabile u. a. 1976] AMABILE, T.M. ; DEJONG, W. ; LEPPER, M.R.: Effects of externally imposed deadlines on subsequent intrinsic motivation. In: *Journal of Personality and Social Psychology* (1976), Nr. 34, S. 92–98

[Asendorpf 1996] ASENDORPF, Jens B.: *Psychologie der Persönlichkeit: Grundlagen*. Berlin : Springer-Verlag, 1996

[Backes-Gellner u. a. 2001] BACKES-GELLNER, Uschi ; LAZEAR, Edward P. ; WOLFF, Birgitta: *Personalökonomik: Fortgeschrittene Anwendungen für das Management*. Stuttgart : Schäffer-Poeschel, 2001

[Bandilla und Hauptmanns 1999] BANDILLA, Wolfgang ; HAUPTMANNS, Peter: Internetbasierte Umfragen als Datenerhebungstechnik für die empirische Sozialfoschung? In: ZENTRUM FÜR UMFRAGEN, METHODEN UND ANALYSEN (ZUMA) (Hrsg.): *ZUMA-Nachrichten 43*. Mannheim : ZUMA, 1999

[Barnett 2004] BARNETT, Liz: Applying Open Source Processes In Corporate Development Organizations. In: *<http://vasoftware.com/sourceforge/request_info-dl.php?paper=9> (Zugriff 7.10.2004)* (2004)

[Batinic 2001] BATINIC, Bernad: *Fragebogenuntersuchungen im Internet: Dissertation Universität Erlangen-Nürnberg*. Aachen : Shaker Verlag, 2001

[Benkler 2002] BENKLER, Yochai: Coase's Penguin, or, Linux and The Nature of the Firm. In: *The Yale Law Journal* 112 (2002), S. 79

[Bernard 1994] BERNARD, H. R.: *Research Methods in Anthropology: Qualitative and Quantitative Approaches*. Walnut Creek : Altamira Press, 1994

[Bessen und Maskin 2000] BESSEN, J. ; MASKIN, E.: *Sequential innovation, patents and imitation*. Cambridge, MA : MIT Economic Departement Working Paper, 2000

[Bessen 2001] BESSEN, James: Open Source Software: Free Provision of Complex Public Goods. In: *<http://www.researchoninnovation.org/opensrc.pdf> (Zugriff 1.10.2001)* (2001)

[Bittman und Goodin 1998] BITTMAN, Michael ; GOODIN, Robert E.: An Equivalence Scale for Time. In: *SPRC Discussion Paper* (1998), Nr. 85, S. 21

[Bitzer u. a. 2004] BITZER, Jürgen ; SCHRETTL, Wolfram ; SCHRÖDER, Philipp J.: Intrinsic Motivation in Open Source Software Development. In: <http://opensource.mit.edu/papers/bitzerschrettlschroder.pdf> *(Zugriff 1.11.2004)* (2004)

[Bland und Altman 1997] BLAND, J. M. ; ALTMAN, Douglas G.: Cronbach's alpha. In: *BMJ* 314 (1997), S. 572

[Bonaccorsi und Rossi 2003] BONACCORSI, Andrea ; ROSSI, Cristina: Altruistic individuals, selfish firms?: The structure of motivation in Open Source software. In: <http://firstmonday.org/issues/issue9_1/bonaccorsi/index.html> *(Zugriff 1.11.2003)* (2003)

[Booth u. a. 1995] BOOTH, Wayne C. ; COLOMB, Gregory G. ; WILLIAMS, Joseph M.: *The Craft of Research*. Chicago : University of Chicago Press, 1995

[Bosnjak und Batinic 1999] BOSNJAK, M. ; BATINIC, B.: Determinanten der Teilnahmebereitschaft an internet-basierten Fragebogenuntersuchungen am Beispiel E-Mail. In: BATINIC, B. (Hrsg.) ; WERNER, A. (Hrsg.) ; GRÄF, L. (Hrsg.) ; BANDILLA, W. (Hrsg.): *Online Research - Methoden, Anwendungen und Ergebnisse*. Göttingen : Hogrefe, 1999

[Broadwell 2005] BROADWELL, Geoff: -Ofun. In: <http://www.oreillynet.com/pub/wlg/7996> *(Zugriff 10.10.2005)* (2005)

[Brooks 1995] BROOKS, Frederick P.: *The Mythical Man-Month: Essays on Software Engineering*. Reading, Massachusetts : Addison-Wesley, 1995

[Challet und Du 2003] CHALLET, Damien ; DU, Yann L.: Closed source versus open source in a model of software bug dynamics. In: <http://www.arxiv.org/abs/cond-mat/0306511> *(Zugriff 25.7.2003)* (2003)

[Chen 2002] CHEN, Hsiang: Exploring Web Users' On-line Positive Affects. (2002)

[Comino und Manetti 2003] COMINO, Stefano ; MANETTI, Fabio M.: Open Source vs Closed Source Software: Public Policies in the Software Market. In: <http://opensource.mit.edu/papers/cominomanenti.pdf> *(Zugriff 16.12.2004)* (2003)

[Crowston u. a. 2005] CROWSTON, Kevin ; HECKMAN, Robert ; ANNABI, Hala ; MASANGO, Chengetai: A structurational perspective on leadership in Free/Libre Open Source Software teams. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Csikszentmihalyi 1975] CSIKSZENTMIHALYI, M.: *Beyond Boredom and Anxiety*. San Francisco : Jossey-Bass, 1975

[Csikszentmihalyi 1990] CSIKSZENTMIHALYI, M.: *Flow: The Psychology of Optimal Experience*. New York : Harper and Row, 1990

[Csikszentmihalyi 1985] CSIKSZENTMIHALYI, Mihaly: *Das flow-Erlebnis: Jenseits von Angst und Langeweile: im Tun aufgehen*. Stuttgart : Klett-Cotta, 1985

[Csikszentmihalyi 2000] CSIKSZENTMIHALYI, Mihaly: *Dem Sinn des Lebens eine Zukunft geben*. Stuttgart : Klett Cotta, 2000

[Csikszentmihalyi 2004] CSIKSZENTMIHALYI, Mihaly: *Flow im Beruf: Das Geheimnis des Glücks am Arbeitsplatz*. Stuttgart : Klett Cotta, 2004

[Csikszentmihalyi und Csikszentmihalyi 1988] CSIKSZENTMIHALYI, Mihaly ; CSIKSZENTMIHALYI, Isabella S.: *Optimal experience: Psychological studies of flow in consciousness*. Cambridge : Cambridge University Press, 1988

[Csikszentmihalyi und Csikszentmihalyi 1991] CSIKSZENTMIHALYI, Mihaly ; CSIKSZENTMIHALYI, Isabella S.: *Die aussergewöhnliche Erfahrung im Alltag: Die Psychologie des Flow-Erlebnisses*. Stuttgart : Klett-Cotta, 1991

[Csikszentmihalyi und LeFevre 1989] CSIKSZENTMIHALYI, Mihaly ; LEFEVRE, Judith: Optimal Experience in Work and Leisure. In: *Journal of Personality and Social Psychology* 56 (1989), Nr. 5, S. 815–822

[Csikszentmihalyi u. a. 1993] CSIKSZENTMIHALYI, Mihaly ; RATHUNDE, Kevin ; WHALEN, Samuel: *Talented teenagers: The roots of success and failure*. New York : Cambridge University Press, 1993

[von Cube 1998] CUBE, Felix von: *Lust an Leistung: Die Naturgesetze der Führung*. München : Piper, 1998

[Dafermos 2001] DAFERMOS, George N.: Management & Virtual Decentralised Networks: The Linux Project. In: *<http://opensource.mit.edu/papers/dafermoslinux.pdf> (Zugriff 21.2.2002)* (2001)

[DeMarco und Lister 1987] DEMARCO, Tom ; LISTER, Timothy: *Peopleware: Productive Projects and Teams*. New York : Dorset House Publishing, 1987

[Dempsey u. a. 2002] DEMPSEY, Bert J. ; WEISS, Debra ; JONES, Paul ; GREENBERG, Jane: Who is an Open Source Software Developer? In: *Communications of the ACM* 45 (2002), Nr. 2, S. 67–73

[Easton u. a. 1997] EASTON, A. N. ; PRICE, J.H. ; TELLJOHANN, S.K. ; BOEHM, K.: An informational versus monetary incentive in increasing physicians' response rates. In: *Psychological Reports* 81 (1997), Nr. 3, S. 968–970

[Eckert u. a. 2005] ECKERT, Daniel ; KOCH, Stefan ; MITLÖHNER, Johann: Using the Iterated Prisoner's Dilemma for Explaining the Evolution of Cooperation in Open Source Communities. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Feller und Fitzgerald 2002] FELLER, Joseph ; FITZGERALD, Brian: *Understanding Open Source Software Development*. London : Addison-Wesley, 2002

[Franck und Jungwirth 2002a] FRANCK, Egon ; JUNGWIRTH, Carola: Das Open-Source-Phänomen jenseits des Gift-Society-Mythos. In: *Wirtschaftswissenschaftliches Studium (WiSt)* 31 (2002), S. 124–129

[Franck und Jungwirth 2002b] FRANCK, Egon ; JUNGWIRTH, Carola: Reconciling rent-seekers and donators. In: *Journal of Management and Governance* 7 (2002), S. 401–421

[Franck und Jungwirth 2003] FRANCK, Egon ; JUNGWIRTH, Carola: Die Governance von Open-Source-Projekten. In: *Zeitschrift für Betriebswirtschaft (ZfB)* 73 (2003), Nr. 5, S. 1–21

[Franck u. a. 2005] FRANCK, Egon ; JUNGWIRTH, Carola ; LUTHIGER, Benno: Motivation und Engagement beim OSS-Programmieren - Eine empirische Analyse. In: *<http://www.isu.unizh.ch/fuehrung/Dokumente/WorkingPaper/36full.pdf> (Zugriff 18.7.2005)* (2005)

[Franke und von Hippel 2002] FRANKE, Nikolaus ; HIPPEL, Eric von: Satisfying Heterogenous User Needs via Innovation Toolkits: The Case of Apache Security Software. In: *<http://opensource.mit.edu/papers/frankevonhippel> (Zugriff 21.2.2002)* (2002)

[Gabriel und Goldman 2002] GABRIEL, Richard P. ; GOLDMAN, Ron: Open Source: Beyond the Fairytales. In: *<http://opensource.mit.edu/papers/gabrielgoldman.pdf> (Zugriff 4.10.2003)* (2002)

[Ghani u. a. 1991] GHANI, Jawaid A. ; SUPNICK, Roberta ; ROONEY, Pamela: The Experience of Flow in Computer-Mediated and in Face-to-Face Groups. In: DEGROSS, J.I (Hrsg.) ; BENBASAT, I. (Hrsg.) ; DESANCTIS, G. (Hrsg.) ; BEATH, C. M. (Hrsg.): *Proceedings of the Twelfth International Conference on Information Systems*. New York, 1991

[Ghosh u. a. 2002] GHOSH, Rishab A. ; GLOTT, Ruediger ; KRIEGER, Bernhard ; ROBLES, Gregorio: FLOSS: Free/Libre and Open Source Software: Survey and Study. In: *<http://www.infonomics.nl/FLOSS/report/> (Zugriff 10.8.2002)* (2002)

[Ghosh 1998] GHOSH, Rishab A.: FM Interview with Linus Torvalds: What motivates free software developers? In: *<http://www.firstmonday.dk/issues/issue3_3/torvalds/index.html> (Zugriff 20.2.2002)* (1998)

[González 2005] GONZÁLEZ, Andrés G.: The calm before the storm? Legal challenges to open source licences. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Gutsche 2005] GUTSCHE, Joerg: Competition between Open Source and Propretiary Software, and the Scope for Public Policy. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Hahn 2002] HAHN, R.W.: *Government Policy toward Open Source Software*. Washington D.C. : AEIBrooking Joint Center for Regulatory Studies, 2002

[Hansmann 1980] HANSMANN, H. B.: The role of nonprofit enterprise. In: *Yale Law Journal* 89 (1980), S. 835–901

[Hars und Ou 2001] HARS, Alexander ; OU, Shaosong: Working for Free? - Motivations of Participating in Open Source Projects. In: *34th Annual Hawaii International Conference on System Sciences* 7 (2001), S. 1–7

[Hayes und Cai 2004] HAYES, Andrew F. ; CAI, Li: Using Heteroscedasticity-Consistent Standard Error Estimators in OLS Regression with Applications to Moderated Multiple Regression. In: *<http://www.jcomm.ohio-state.edu/ahayes/hcse.pdf> (Zugriff 20.1.2005)* (2004)

[Healy und Schussman 2003] HEALY, Kieran ; SCHUSSMAN, Alan: The Ecology of Open Source Software Development. In: *<http://opensource.mit.edu/papers/healyschussman.pdf> (Zugriff 30.1.2003)* (2003)

[Hecker 1999] HECKER, Frank: Setting Up Shop: The Business of Open-Source Software. In: *IEEE Software* 16 (1999), Nr. 1, S. 45–51

[Hemetsberger 2003] HEMETSBERGER, Andrea: When Consumers Produce on the Internet: The Relationship between Cognitive-affective, Socially-based, and Behavioral Involvement of Prosumers. In: *<http://opensource.mit.edu/papers/hemetsberger1.pdf> (Zugriff 29.7.2004)* (2003)

[Hertel u. a. 2003] HERTEL, Guido ; NIEDNER, Sven ; HERRMANN, Stefanie: Motivation of Software Developers in Open Source Projects. In: *Research Policy* 32 (2003), Nr. 7, S. 1159–1177

[von Hippel und von Krogh 2002] HIPPEL, Eric von ; KROGH, Georg von: Exploring the Open Source Software Phenomenon: Issues for Organization Science. (2002)

[Hoffman und Novak 1996] HOFFMAN, Donna L. ; NOVAK, Thomas P.: Marketing in Hypermedia Computer-Mediated Environments: Conceptual Foundations. In: *Journal of Marketing* 60 (1996), S. 50–68

[Howison u. a. 2005] HOWISON, James ; CONKLIN, Megan ; CROWSTON, Kevin: OSSmole: A collaborative repository for FLOSS research data and analyses. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Howison und Crowston 2004] HOWISON, James ; CROWSTON, Kevin: The perils and pitfalls of mining SourceForge. In: *<http://opensource.mit.edu/papers/howison04msr.pdf> (Zugriff 25.11.2004)* (2004)

[Huizinga 2001] HUIZINGA, Johan: *Homo Ludens: Vom Ursprung der Kultur im Spiel*. Hamburg : Rowohlt, 2001

[Johnson 2001] JOHNSON, Justin P.: Economics of Open Source Software. (2001)

[Jorgensen 2001] JORGENSEN, Niels: Putting it All in the Trunk. In: *Information Systems Journal* 11 (2001), Nr. 4

[Kim 2003] KIM, Eugene E.: An Introduction to Open Source Communities. In: *<http://www.blueoxen.org/> (Zugriff 20.4.2003)* (2003)

[Kollock 1999] KOLLOCK, Peter: The Economics of Online Cooperation: Gifts and Public Good in Cyberspace. In: SMITH, Marc A. (Hrsg.) ; KOLLOCK, Peter (Hrsg.): *Communities in Cyberspace*. London : Routledge, 1999

[Kollock und Smith 1996] KOLLOCK, Peter ; SMITH, Marc: Managing the Virtual Commons: Cooperation and Conflict in Computer Communities. In: HERRING, Susan (Hrsg.): *Computer-Mediated Communication: Linguistic, Social, and Cross-Cultural Perspectives*. Amsterdam : John Benjamins, 1996

[Krishnamurthy 2002] KRISHNAMURTHY, Sandeep: Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects. In: *<http://www.firstmonday.org/issues/issue7_6/krishnamurthy/index.html> (Zugriff 6.6.2002)* (2002)

[Kubey und Csikszentmihalyi 1996] KUBEY, Robert ; CSIKSZENTMIHALYI, Mihaly: Experience Sampling Method. In: *Journal of Communication* 46 (1996), Nr. 2, S. 99–120

[Kuwabara 2000] KUWABARA, Ko: Linux: A Bazaar at the Edge of Chaos. In: *First Monday* 5 (2000), Nr. 3, S. 1–73

[Lakhani und von Hippel 2000] LAKHANI, Karim ; HIPPEL, Eric von: How Open Source software works: „Free“ user-to-user assistance. In: *<http://web.mit.edu/evhippel/www/opensource.PDF> (Zugriff 19.9.2001)* (2000)

[Lakhani und Wolf 2003] LAKHANI, Karim R. ; WOLF, Robert G.: Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects. In: *<http://opensource.mit.edu/papers/lakhaniwolf.pdf> (Zugriff 6.10.2003)* (2003)

[Leiteritz 2004] LEITERITZ, Raphael: Open Source-Geschäftsmodelle. In: GEHRING, Robert A. (Hrsg.) ; LUTTERBECK, Bernd (Hrsg.): *Open Source Jahrbuch 2004*. Berlin : Lehmanns Media, 2004

[Lerner und Tirole 2001] LERNER, Josh ; TIROLE, Jean: The Simple Economics of Open Source. In: *<http://opensource.mit.edu/papers/> (Zugriff 19.7.2001)* (2001)

[Lerner und Tirole 2002] LERNER, Josh ; TIROLE, Jean: The Scope of Open Source Licensing. (2002)

[Lorenz und Wagner 1988] LORENZ, Wilhelm ; WAGNER, Joachim: Kompensierende Lohndifferentiale. In: *WiSt-Wirtschaftswissenschaftliches Studium* 17 (1988), Nr. 10, S. 515–518

[Luthiger 2003] LUTHIGER, Benno: Wie relevant ist die Open Source-Szene Schweiz? (2003)

[Luthiger 2004] LUTHIGER, Benno: Alles aus Spass? In: GEHRING, Robert A. (Hrsg.) ; LUTTERBECK, Bernd (Hrsg.): *Open Source Jahrbuch 2004*. Berlin : Lehmanns Media, 2004

[Luthiger 2005] LUTHIGER, Benno: Fun and Software Development. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Markt und e.V. (ADM) 2000] MARKT, Arbeitskreis D. ; (ADM), Sozialforschungsinstitute e.V.: Richtlinie für Online-Befragungen. In: *<http://www.adm-ev.de/pdf/R_08D.PDF> (Zugriff 10.2.2003)* (2000)

[Marr 1998] MARR, Arthur J.: The Flow Experience: Or What Occurs When Bad Science Happens to Good Observations. (1998)

[Montgomery u. a. 2004] MONTGOMERY, Henry ; SHARAFI, Parvaneh ; HEDMAN, Leif R.: Engaging in Activities Involving Information Technology: Dimensions, Modes, and Flow. In: *Human Factors* 46 (2004), Nr. 2, S. 334–348

[Moon und Sproull 2000] MOON, Jae Y. ; SPROULL, Lee: Essence of Distributed Work: The Case of the Linux Kernel. In: *<http://www.firstmonday.org/issues/issue5_11/moon/index.html> (Zugriff 7.3.2002)* (2000)

[Novak und Hoffman 1997] NOVAK, Thomas P. ; HOFFMAN, Donna L.: Measuring the Flow Experience Among Web Users. In: *<http://www2000.ogsm.vanderbilt.edu/> (Zugriff 10.10.2002)* (1997)

[Novak und Hoffman 1999] NOVAK, Thomas P. ; HOFFMAN, Donna L.: Measuring the Flow Construc in Online Environments: A Structural Modeling Approach. In: *<http://www2000.ogsm.vanderbilt.edu/> (Zugriff 10.10.2002)* (1999)

[Novak u. a. 1997] NOVAK, Thomas P. ; HOFFMAN, Donna L. ; YUNG, Yiu-Fai: Modeling the Structure of the Flow Experience Among Web Users. In: *<http://www2000.ogsm.vanderbilt.edu/> (Zugriff 10.10.2002)* (1997)

[Novak u. a. 1998] NOVAK, Thomas P. ; HOFFMAN, Donna L. ; YUNG, Yiu-Fai: Measuring the Flow Construc in Online Environments: A Structural Modeling Approach. In: *<http://www2000.ogsm.vanderbilt.edu/> (Zugriff 10.10.2002)* (1998)

[Open Source Initiative 1999] OPEN SOURCE INITIATIVE: The Open Source Definition: Version 1.9. In: *<http://www.opensource.org/docs/definition_plain.html> (Zugriff 12.06.2005)* (1999)

[Osterloh und Rota 2002] OSTERLOH, Margit ; ROTA, Sandra: Open Source Software: New Rules for the Market Economy? (2002)

[Osterloh u. a. 2002a] OSTERLOH, Margit ; ROTA, Sandra ; KUSTER, Bernhard: Open Source Software Production: Climbing on the Shoulders of Giants. (2002)

[Osterloh u. a. 2002b] OSTERLOH, Margit ; ROTA, Sandra ; KUSTER, Bernhard: Trust and Commerce in Open Source: a Contradiction? In: *<http://www.unizh.ch/ifbf/orga/downloads/publikationen/osterloh_rota_kuster.pdf> (Zugriff 4.12.2002)* (2002)

[Porst 1996] PORST, Rolf: Fragebogenerstellung. In: GOEBL, H. (Hrsg.) ; NELDE, P. H. (Hrsg.) ; STARÝ, Z. (Hrsg.) ; WÖLCK, W. (Hrsg.): *Kontaktlinguistik - Contact Linguistics - Linguistique de contact*. Berlin und New York : De Gruyter, 1996

[Porst 1998] PORST, Rolf: Im Vorfeld der Befragung: Planung, Fragebogenentwicklung, Pretesting. In: *ZUMA-Arbeitsbericht 98/02* 98 (1998), Nr. 2, S. 1–45

[Porst u. a. 1998] PORST, Rolf ; RANFT, Sabine ; RUOFF, Bernd: Strategien und Massnahmen zur Erhöhung der Ausschöpfungsquoten bei sozialwissenschaftlichen Umfragen. In: *ZUMA-Arbeitsbericht 98/07* 98 (1998), Nr. 7, S. 1–25

[Prüfer und Rexroth 1996] PRÜFER, Peter ; REXROTH, Margrit: Verfahren zur Evaluation von Survey - Fragen: Ein Überblick. In: *ZUMA-Arbeitsbericht* 96 (1996), Nr. 5, S. 1–39

[Privette und Bundrick 1987] PRIVETTE, G. ; BUNDRICK, Charles M.: Measurement of Experience: Constructs and Content Validity of the Experience Questionnaire. In: *Perceptual and motor skills* 65 (1987), S. 315 – 332

[Raymond 1999a] RAYMOND, Eric S.: *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol : O'Reilly, 1999

[Raymond 1999b] RAYMOND, Eric S.: The Magic Cauldron. In: *<http://www.catb.org/~esr/writings/magic-cauldron/> (Zugriff 21.7.2003)* (1999)

[Raymond 2000a] RAYMOND, Eric S.: Homesteading the Noosphere. In: *<http://www.catb.org/~esr/writings/homesteading/> (Zugriff 21.7.2003)* (2000)

[Raymond 2000b] RAYMOND, Eric S.: The Cathedral and the Bazaar. In: *<http://www.catb.org/~esr/writings/cathedral-bazaar/> (Zugriff 21.7.2003)* (2000)

[Remy 2002] REMY, Katharina: *Entwicklung eines Fragebogens zum Flow-Erleben*. Bielefeld: Diplomarbeit : Fakultät für Psychologie und Sportwissenschaft, 2002

[Rheinberg 1997] RHEINBERG, Falko: *Motivation*. Stuttgart : Kohlhammer, 1997

[Rheinberg u. a. 2002] RHEINBERG, Falko ; VOLLMEYER, Regina ; ENGESER, Stefan: Die Erfassung des Flow-Erlebens. In: STIENSMEIER-PELSTER, Joachim (Hrsg.) ; RHEINBERG, Falko (Hrsg.): *Diagnostik von Motivation und Selbstkonzept*. Göttingen : Hogrefe, 2002

[Robles u. a. 2005] ROBLES, Gregorio ; GONZALEZ-BARAHON, Jesus M. ; MICHLMAYR, Martin: Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Robles u. a. 2001] ROBLES, Gregorio ; SCHEIDER, Hendrik ; TRETKOWSKI, Ingo ; WEBER, Niels: Who is doing it?: A research on Libre Software developers. In: *<http://widi.berlios.de/paper/study.html> (Zugriff 18.11.2002)* (2001)

[Rossi 2004] ROSSI, Maria A.: Decoding the „Free/Open Source (F/OSS) Puzzle“ - a Survey of Theoretical and Empirical Contributions. In: *<http://opensource.mit.edu/papers/rossi.pdf> (Zugriff 25.11.2004)* (2004)

[Scacchi 2001] SCACCHI, Walt: Understanding the Requirements for Developing Open Source Software Systems. In: *<http://opensource.mit.edu/papers/Scacchi.pdf> (Zugriff 6.3.2002)* (2001)

[Scacchi u. a. 2005] SCACCHI, Walt ; JENSEN, Chris ; NOLL, John ; ELLIOTT, Margaret: Multi-Modal Modeling of Open Source Software Requirements Processes. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Schallberger und Pfister 2003] SCHALLBERGER, Urs ; PFISTER, Regula: Flow-Erleben in Arbeit und Freizeit. In: *Zeitschrift für Arbeits- und Organisationspsychologie* (2003), Nr. 45, S. 176–187

[Schäfer 2005] SCHÄFER, Annette: Mr Flow und die Suche nach dem Guten Leben. In: *Psychologie Heute* 32 (2005), Nr. 3, S. 43–48

[Schmidt und Schnitzer 2003] SCHMIDT, Klaus M. ; SCHNITZER, Monika: Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market. In: *Harvard Journal of Law and Technology* 16 (2003), Nr. 2, S. 473–505

[Scott Card 1995] SCOTT CARD, Orson: How Software Companies Die. In: *<http://www.netjeff.com/humor/item.cgi?file=DeveloperBees> (Zugriff 1.7.2005)* (1995)

[Sieckmann 2001] SIECKMANN, Jens: Bravehack: Technische, wirtschaftliche und gesellschaftliche Aspekte von freier Software und Open Source. In: *<http://www.bravehack.de/html/> (Zugriff 1.12.2001)* (2001)

[Spaeth 2005] SPAETH, Sebastian: *Coordination in Open Source Projects: A Social Network Analysis using CVS data*. Bamberg : Difo-Druck GmbH, 2005

[Toffler 1981] TOFFLER, Alvin: *The Third Wave*. New York, 1981

[Torvalds und Diamond 2001] TORVALDS, Linus ; DIAMOND, David: *Just for FUN: The Story of an Accidental Revolutionary*. New York : Harper Collins Publishers, 2001

[Wagstrom u. a. 2005] WAGSTROM, Patrick A. ; HERBSLEB, James D. ; CARLEY, Kathleen: A Social Network Approach to Free/Open Source Software Simulation. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Weber 2000] WEBER, Steven: The Political Economy of Open Source Software. In: *<http://brie.berkeley.edu/~briewww/pubs/wp/wp140.pdf> (Zugriff 6.3.2002)* (2000)

[Webster und Martocchio 1992] WEBSTER, Jane ; MARTOCCHIO, Joseph J.: Microcomputer Playfulness: Development of a Measure With Workplace Implications. In: *MIS Quarterly* 16 (1992), Nr. 6, S. 201–26

[Webster u. a. 1993] WEBSTER, Jane ; TREVINO, Linda K. ; RYAN, Lisa: The Dimensionality and Correlates of Flow in Human Computer Interactions. In: *Computers in Human Behavior* 9 (1993), Nr. 4, S. 411–26

[Weiss 2005] WEISS, Dawid: Quantitative Analysis of Open Source Projects on SourceForge. In: SCOTTO, Marco (Hrsg.) ; SUCCI, Giancarlo (Hrsg.): *Proceedings of the 1st International Conference on Open Source Systems*. Genova : ECIG, 2005

[Wiebe 2004] WIEBE, Andreas: Patentschutz und Softwareentwicklung - ein unüberbrückbarer Gegensatz? In: GEHRING, Robert A. (Hrsg.) ; LUTTERBECK, Bernd (Hrsg.): *Open Source Jahrbuch*. Berlin : Lehmanns Media, 2004

[Zeitlyn 2003] ZEITLYN, David: Gift economies in the development of open source software: anthropological reflections. In: *Research Policy* 32 (2003), Nr. 2003, S. 1287–1291

Curriculum Vitae

Personalien

Name:	Benno Luthiger Stoll
Geboren:	24. Dez. 1961 in Luzern

Ausbildung

1968 - 1974	Primarschule in Luzern
1974 - 1981	Kantonsschule Luzern: Realgymnasium mit Matura Typ C
1981 - 1988	Physikstudium an der Universität Zürich
1989 - 1997	Studium in Ethnologie und Ökonomie an der Universität Zürich
2002 - 2005	Doktorandenstudium an der Wirtschaftswissenschaftlichen Fakultät der Universität Zürich

Berufliche Tätigkeiten

1989 - 1995	Programmierer beim Betriebsdienst der ETH Zürich
1996	6 Monate Feldforschungsaufenthalt in Uganda im Rahmen des NFS-Projekts „Use and Protection of Water Resources in Lake Victoria through Sustainable Management of Wetland-Ecotones“ (Datenaufnahme für Lizentiats-Arbeit in Ökonomie)
1996 - 1999	Programmierer bei den Informatikdiensten (Sektion Administrative Informatik) der ETH Zürich
1999 - 2002	Tätigkeit als Application-Engineer und Technischer Projektleiter bei der Systor AG
seit 2002	Anstellung als Fachperson für Open Source an der ETH Zürich: Unterstützung der Gruppe „Technologie- und Informationsmanagement“ der Informatikdienste bei der Einführung von Open-Source-Applikationen